ALIBABA CLOUD

Alibaba Cloud

云数据库 Redis 版

产品简介

文档版本：20220712

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.什么是云数据库Redis版

云数据库Redis版（ApsaraDB for Redis）是兼容开源Redis协议标准、提供混合存储的数据库服务，基于双机热备架构及集群架构，可满足高吞吐、低延迟及弹性变配等业务需求。

为什么选择云数据库Redis版

- 硬件部署在云端，提供完善的基础设施规划、网络安全保障和系统维护服务，您可以专注于业务创新。
- 支持String（字符串）、List（链表）、Set（集合）、Sorted Set（有序集合）、Hash（哈希表）、Stream（流数据）等多种数据结构，同时支持Transaction（事务）、Pub/Sub（消息订阅与发布）等高级功能。
- 在社区版的基础上推出企业级内存数据库产品，提供性能增强型、持久内存型、容量存储型和混合存储型（已停售）供您选择。

更多详情请参见云数据库Redis版与自建Redis的对比和应用场景。

购买方式

创建Redis实例

实例类型

版本类型	简介
云数据库Redis社区版	兼容开源Redis的高性能内存数据库产品，支持主从双副本、集群、读写分离等架构。
云数据库Redis企业版	Redis企业版作为云数据库Redis社区版的基础上开发的强化版Redis服务，从访问延时、持久化需求、整体成本这三个核心维度考量，基于DRAM、NVM和ESSD云盘等存储介质，推出了多种不同形态的产品，为您提供更强的性能、更多的数据结构和更灵活的存储方式，满足不同场景下的业务需求。 • 性能增强型：采用多线程模型，集成阿里巴巴Tair的部分特性，支持多种Tair数据结构，对于部分特殊业务有很高的适用性。 • 持久内存型：基于Intel傲腾™持久内存，为您提供大容量、兼容Redis的内存数据库产品。数据持久化不依赖传统磁盘，保证每个操作持久化的同时提供近乎Redis社区版的吞吐和延时，极大提升业务数据可靠性。 • 容量存储型：基于云盘ESSD研发，兼容Redis核心数据结构与接口，可提供大容量、低成本、强持久化的数据库服务，适用于兼容Redis、需要大容量且较高访问性能的温冷数据存储场景。

实例架构

云数据库Redis版支持灵活的多种部署架构，能够满足不同的业务场景。

架构类型	说明
标准版-双副本	系统工作时主节点（Master）和副本（Replica）数据实时同步，若主节点发生故障，系统会快速将业务切换至备节点，全程自动且对业务无影响，保障服务高可用性。
集群版-双副本	集群（Cluster）实例采用分布式架构，每个数据分片都支持主从切换（master-replica），能够自动进行容灾切换和故障迁移，保障服务高可用。同时提供多种规格，您可以根据业务压力选择对应规格，还可以随着业务的发展自由变配规格。集群版支持两种连接模式： • 代理模式是集群版的默认连接方式，可提供智能的连接管理，降低应用开发成本。 • 直连模式支持客户端绕过代理服务器直接访问后端数据分片，可降低网络开销和服务响应时间，适用于对Redis响应速度要求极高的业务。
Redis读写分离版	与标准版-双副本架构类似，读写分离实例采用主从（Master-Replica）架构提供高可用，主节点挂载只读副本（Read Replica）实现数据复制，支持读性能线性扩展。 只读副本可以有效缓解热点key带来的性能问题，适合高读写比的业务场景。 读写分离实例提供非集群版和集群版： • 非集群版读写分离实例支持一个只读副本、三个只读副本或五个只读副本三种版本。 • 集群版读写分离实例在每个分片下挂载一个只读副本，提供分片级别的自动读写分离能力，适合超大规模高读写比的业务场景。

实例规格

云数据库Redis社区版和企业版的每种架构都有多种规格，详细信息请参见规格查询导航。

2.云数据库Redis初识与选型

2.1. 阿里云Redis产品选型必读

创建Redis实例前，您需要结合产品性能、价格、业务场景、工作负载等因素，做出性价比与稳定性最优的决策。本文围绕以上因素，着重介绍实例类型、引擎版本、架构、存储介质，为您的选型提供相关参考。

了解云数据库Redis产品系列

云数据库Redis版（ApsaraDB for Redis）是兼容开源Redis协议、提供丰富存储介质的数据库服务，基于双机热备架构及集群架构，可满足高吞吐、低延迟及弹性变配等业务需求。

选型流程

通常，您需要结合产品性能、价格、业务场景（例如用作还是）、工作负载等因素，选择实例的类型与规格，推荐的选型流程如下：高速缓存内存数据库

> **说明** 关于费用的相关信息，请参见收费项与价格。

选型操作	说明
选择社区版或企业版	云数据库Redis在提供社区版的同时，还基于阿里云内部使用的Tair产品研发并推出强化版Redis服务（Redis企业版），为您提供更强的性能、更多的数据结构和更灵活的存储方式。
选择部署架构	云数据库Redis支持标准架构、集群架构和读写分离架构，可满足不同的业务场景对业务读写能力、数据量和性能的要求。
选择容灾方案	当云数据库Redis实例因不可预料的原因（例如设备故障、机房断电等）发生故障，容灾机制可用于保障数据的一致性和业务可用性。云数据库Redis提供多种灾备方案供您选择，可满足不同的业务场景。
选择大版本	推荐使用更新的大版本以支持更多的功能和特性。
预估内存规格	提前预估可能消耗的内存容量，可以帮助节约成本、避免频繁变更规格给业务带来的影响。
创建Redis实例	完成上述实例的选型后，您可以通过控制台或调用OpenAPI创建Redis实例。
服务能力验证与调整	当您完成选型并开始使用Redis实例后，您需要观察业务正常运行状态下的性能监控信息，验证当前实例的服务能力是否符合预期。

选择云盘版或本地盘版

当前云数据库Redis的本地盘实例功能较为完整，云盘版实例为云原生基础架构，支持集群无感扩缩容，可通过自定义分片数量的方式来实现扩缩容，未来会以云盘版为主要演进方向，详细对比如下。

对比项	本地盘实例	云盘实例
架构	基于云数据库Redis传统管控架构。	基于云数据库Redis新一代管控架构。 **说明** 后续的产品将基于此架构演进，同时将会提供本地盘实例到云盘实例的迁移功能。
扩容能力	• 扩容耗时较长。 • 集群架构的实例扩容会有闪断。 • 集群架构实例的分片节点的扩展数固定，例如2分片、4片、8分片等。	• 扩容能力比本地盘更好。 • 集群架构的实例扩容无闪断。 • 集群架构的实例支持自由调整分片节点的数量（最少1个分片节点），支持单分片的扩缩容，可更好地应对读写热点和倾斜。

选择社区版或企业版

云数据库Redis在提供社区版的同时，还基于阿里云内部使用的Tair产品研发并推出企业级内存数据库产品，即Redis企业版（Tair）。Redis企业版从访问延时、持久化需求、整体成本这三个核心维度考量，基于DRAM、NVM和ESSD云盘存储介质，推出了多种系列，为您提供更强的性能、更多的数据结构和更灵活的存储方式，满足不同场景下的业务需求。

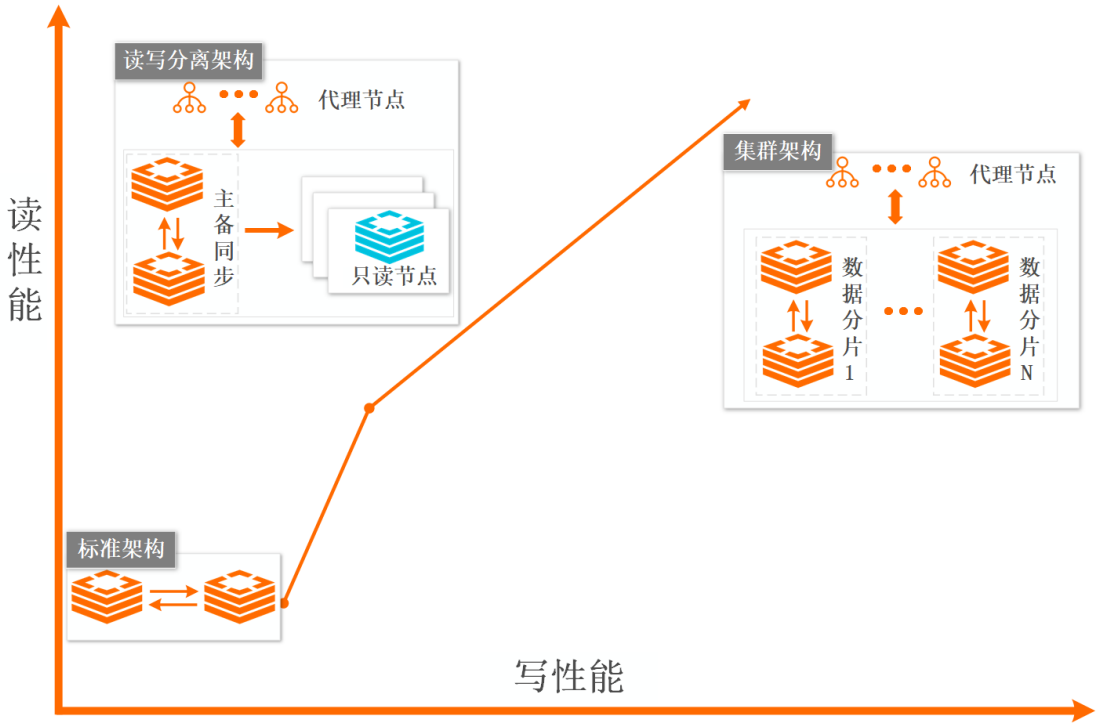
> 注意
> - Redis企业版在兼容社区版的基础上，还支持了一些高级特性（例如通过数据闪回按时间点恢复数据、代理查询缓存、全球多活等），关于社区版和企业版的能力和性能的详细对比，请参见特性对比。
> - 各系列支持的命令与参数，请参见Redis命令支持概览和设置实例参数。

类别	系列	特点	适用场景
Redis企业版	性能增强型	• 采用多线程模型：性能约为同规格社区版实例的3倍。 • 支持多种增强型数据结构模块（modules），包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，使业务无需再关心存储的结构和时效性，能够极大提升业务开发效率。 • 超高兼容性：100%兼容原生Redis，无需修改业务代码。 • 支持诸多企业级特性：通过数据闪回按时间点恢复数据、代理查询缓存、全球多活等。	以性能为中心的关键业务场景。
	持久内存型	• 超高性价比：相同容量下对比Redis社区版，价格降低30%左右，性能可达原生Redis的90%。 • 支持增强型数据结构模块（modules）：TairString（含CAS和CAD命令）、TairCpc。 • 掉电数据不丢失：强大的命令级持久化保障，每个写操作持久化成功后返回，可将其作为内存数据库（非缓存）使用。 • 大规格优化：解决大规格下执行AOF重写调用fork引起的延时抖动等问题。 • 高兼容性：兼容绝大部分原生Redis的数据结构和命令。	需要高性能且高数据持久化要求，且成本作为次要考虑因素的数据缓存与存储场景。
	容量存储型	• 低成本：最低为Redis社区版的15%。 • 云盘存储：数据分布在ESSD云盘，容量可达百TB级别，拥有高数据可靠性。 • 大规格优化：解决了原生Redis固有的fork问题而预留部分内存的问题。 • 高兼容性：兼容大部分原生Redis的数据结构和命令。	大存储、低访问密度、低访问延迟要求，且成本作为首要考虑因素的数据存储场景。
Redis社区版	无	兼容开源Redis，高性能。	适合中小型或验证型应用，适用于标准化Redis使用和迁移场景。

选择部署架构

云数据库Redis支持三种不同的部署架构，可满足不同的业务场景对业务读写能力、数据量和性能的要求。

部署架构对比



说明

- 集群架构和读写分离架构的实例默认提供代理（Proxy）连接地址，客户端的请求由代理节点转发至数据分片，可享受代理节点带来的负载均衡、读写分离、故障转移、代理查询缓存（仅性能增强型支持）、长连接等特性能力。更多信息，请参见Redis Proxy特性说明。
- 对于集群架构的实例，可申请直连地址，通过该地址可绕过代理，直接访问后端的数据分片（类似连接原生Redis集群）。相比代理模式，直连模式节约了通过代理处理请求的时间，可以在一定程度上提高Redis服务的响应速度。

实例架构	简介	适用场景
标准架构	采用主从（master-replica）双副本架构，由主节点提供日常服务访问，备节点提供高可用。当主节点发生故障，系统会自动在30秒内切换至备节点，保障业务平稳运行。	• 对Redis协议兼容性要求较高的业务。 • 将Redis作为持久化数据存储使用的业务。 • 单个Redis性能压力可控的场景。 • Redis命令相对简单，排序和计算之类的命令较少的场景。
集群架构	• 由代理节点、数据分片和配置服务器组件构成，可通过增加数据分片的方式实现横向扩展。 • 每个数据分片均为双副本（分别部署在不同机器上）高可用架构，主节点发生故障后，系统会自动进行主备切换保证服务高可用。	• 数据量较大的场景。 • 整体读写请求的QPS压力较大的场景。 • 吞吐密集型、高性能应用场景。
读写分离架构	• 由代理节点、主从节点和只读节点构成。 • 只读节点采取链式复制架构，扩展只读节点个数可使整体实例性能呈线性增长。	• 读请求QPS压力较大的场景（如热点数据集中）。 • 对Redis协议兼容性要求较高的业务场景，例如规避集群架构的使用限制。 **说明** 由于数据同步至只读节点存在一定延迟，不适用于数据一致性要求高的场景，可选用集群架构。

选择容灾方案

Redis容灾架构演进

灾备方案	灾备级别	说明
单可用区高可用方案	★★★☆☆	主备节点部署在同一可用区中的不同机器上，当任一节点发生故障时，由高可用HA（High Availability）系统自动执行故障切换，避免单点故障引起的服务中断。
同城容灾方案	★★★★☆	主备节点分别部署在同一地域下两个不同的可用区，当任一可用区因电力、网络等不可抗因素失去通信时，高可用HA系统将执行故障切换，确保整个实例的持续可用。
跨地域容灾方案	★★★★★	由多个子实例构成全球分布式实例，所有子实例通过同步通道保持实时数据同步，由通道管理器负责子实例的健康状态监测、主备切换等等异常事件的处理，适用于异地灾备、异地多活、应用就近访问、分摊负载等场景。更多介绍，请参见全球多活。

选择大版本

可根据业务需求选择大版本（各大版本均长期维护），您也可以使用更新的大版本以支持更多的特性，详情请参见Redis 6.0新特性说明、Redis 5.0新特性说明和Redis 4.0新特性说明。

大版本的选择存在一定的约束，具体如下：

实例及其创建方法	支持的实例类型	支持的引擎版本	支持的架构
本地盘实例 创建Redis本地盘实例社区版或性能增强型实例	Redis社区版	4.0 5.0	集群架构 标准架构 读写分离架构
	企业版（性能增强型）	5.0	集群架构 标准架构 读写分离架构
云盘实例 创建Redis社区版云盘实例	Redis社区版	7.0 6.0 5.0	标准架构 集群架构
	企业版（性能增强型）	5.0	标准架构 集群架构
	企业版（持久内存型）	兼容Redis 6.0	标准架构 集群架构
	企业版（容量存储型）	兼容Redis 4.0	标准架构

预估内存规格

通常情况下，您需要考虑下述因素预估可能消耗的内存容量并在创建实例时选择对应的规格，该操作有助于节约成本、避免频繁变更规格给业务带来的影响，助力业务快速上云。

> 注意 在确定云数据库Redis实例的内存容量时，首先要考虑存储的业务数据大小，除此之外，您还需额外考虑Redis自身运行占用的必要内存开销（例如进程元数据、复制缓冲区、碎片等）。
> 不同于自建Redis数据库，选用云数据库Redis时，您无需再额外考虑云数据库Redis持久化Fork写时复制占用的内存开销以及云数据库Redis增强功能（如安全白名单、审计、大Key、热Key等）的内存开销，这些开销由阿里云承担，不计入购买的实例内存容量。

- Key的数据类型、长度和数量。

> **说明** 如果使用可包含元素的数据类型（例如Hash），您还需要计算每个Key中，各元素的数量和长度。

- Value的长度。
- Key的过期时间与逐出策略。
- 访问模型，例如大量的客户端连接、使用Lua脚本或事务等，均需要为其预留适量的内存。
- 中长期的业务增长情况。

创建Redis实例

完成上述实例的选型后，您可以通过控制台或调用OpenAPI创建Redis实例：

- 控制台：创建Redis实例。
- OpenAPI：CreateInstance。

服务能力验证与调整

云数据库Redis支持非常丰富的监控指标，当您完成选型并开始使用Redis实例后，建议您观察业务正常运行状态下的性能监控信息，验证当前实例的服务能力是否符合预期。具体操作，请参见查看监控数据。

> **说明** 您也可以使用Redis-benchmark执行性能压测进行验证，更多信息，请参见Redis-benchmark使用说明。

例如，当您通过性能监控发现实例内存使用率一直较高，您需要先排查内存使用率较高的原因，如无异常，可升级至更高的规格，具体操作，请参见变更实例配置。关于实例性能类问题的排查方法，请参见：

- 排查Redis实例CPU使用率高的问题
- 排查Redis实例内存使用率高的问题
- 排查Redis实例流量使用率高的问题

相关文档

- 云数据库Redis开发运维规范

2.2. 云数据库Redis版与自建Redis的对比

相比自购服务器搭建Redis数据库，云数据库Redis版在数据安全、运维投入、内核优化等方面都有一定的优势。

对比项	云数据库Redis版	自建Redis
安全防护	事前防护： • VPC网络隔离。 • 白名单控制访问。 • 自定义账号与权限。	事前防护： • 需自行构建网络安全体系，成本高，难度大。 • 社区版Redis的默认访问配置存在安全漏洞，可能导致Redis数据泄露。 • 无账号鉴权体系。
	事中保护：SSL加密访问。	事中保护：需要自行通过第三方工具实现SSL加密访问。
	事后审计：审计日志。	事后审计：无审计功能。
备份恢复	Tair（Redis企业版）性能增强型支持数据闪回功能，可以恢复指定时间点的数据。更多信息，请参见通过数据闪回按时间点恢复数据。	仅支持一次性全量恢复。
运维管理	• 支持十余组监控指标，最小监控粒度为5秒。更多信息，请参见监控指标说明。 • 支持报警设置。 • 可根据需求创建多种架构的实例，支持变配到其它架构和规格。 • 提供基于快照的大key分析功能，精度高，无性能损耗。更多信息，请参见离线全量Key分析。	• 需使用管理方式复杂的第三方监控工具实现服务监控。 • 改变规格或架构的操作复杂，且需要停止服务。 • 支持基于采样的大key分析，统计粗糙，精度较低。
部署和扩容	即时开通，弹性扩容。	需要自行完成采购硬件、机房托管、部署机器等工作，周期较长，且需要自行维护节点关系。

对比项	云数据库Redis版	自建Redis
高可用	• 单可用区高可用方案。 • 同城容灾方案。 • 高可用性由独立的中心化模块保障，决策效率高且稳定，不会出现脑裂（split brain）现象。	• 需要自行部署基于哨兵模式的机房内高可用架构。 • 可基于哨兵模式搭建同城容灾架构。 • 高可用性由哨兵机制保障，搭建成本高，且在业务高峰期决策效率低，可能发生脑裂导致业务受损。
内核优化	• Tair（Redis企业版）提供多线程的增强性能实例，性能为同规格标准版实例的3倍。 • Tair（Redis企业版）提供容量存储型和持久内存型实例，支持大容量存储和命令级别持久化。	• 6.0以上版本支持多IO线程以增强性能，性能至多提升2倍，且CPU资源消耗高。 • 可采用SSDB、Pika等持久化存储方案，但对Redis协议的兼容度低，仅支持key级别冷热数据管理，大key交换成本高，管理困难。
内存	已购内存100%可用，容灾、运维管理、扩容、实例持久化（Fork写时复制）等占用的内存开销均由阿里云承担，不占用实例内存容量。 例如：采购64 GB的云数据库Redis版实例，用户可用内存为64 GB。	需预留25% ~ 40%的内存资源用于容灾、运维管理、扩容等用途。 例如：采购2台内存为64 GB的ECS搭建Redis主从实例，用户可用内存通常低于45 GB。

说明 云数据库Redis版与原生Redis完全兼容（请参见云数据库Redis兼容文档），连接数据库的方式也基本相同，您可以根据自身应用特点选用任何兼容Redis协议的客户端程序，详情请参见通过客户端程序连接Redis。

2.3. Redis企业版与社区版特性对比

本文列出Redis企业版各形态产品与Redis社区版产品的相关特性对比，为您的产品选型提供相关参考。

选型参考

类别	系列	特点	适用场景
Redis企业版	性能增强型	• 采用多线程模型：性能约为同规格社区版实例的3倍。 • 支持多种增强型数据结构模块（modules），包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，使业务无需再关心存储的结构和时效性，能够极大提升业务开发效率。 • 超高兼容性：100%兼容原生Redis，无需修改业务代码。 • 支持诸多企业级特性：通过数据闪回按时间点恢复数据、代理查询缓存、全球多活等。	以性能为中心的关键业务场景。
	持久内存型	• 超高性价比：相同容量下对比Redis社区版，价格降低30%左右，性能可达原生Redis的90%。 • 支持增强型数据结构模块（modules）：TairString（含CAS和CAD命令）、TairCpc。 • 掉电数据不丢失：强大的命令级持久化保障，每个写操作持久化成功后返回，可将其作为内存数据库（非缓存）使用。 • 大规格优化：解决大规格下执行AOF重写调用fork引起的延时抖动等问题。 • 高兼容性：兼容绝大部分原生Redis的数据结构和命令。	需要高性能且高数据持久化要求，且成本作为次要考虑因素的数据缓存与存储场景。

类别	系列	特点	适用场景
	容量存储型	• 低成本：最低为Redis社区版的15%。 • 云盘存储：数据分布在ESSD云盘，容量可达百TB级别，拥有高数据可靠性。 • 大规格优化：解决了原生Redis固有的fork问题而预留部分内存的问题。 • 高兼容性：兼容大部分原生Redis的数据结构和命令。	大存储、低访问密度、低访问延迟要求，且成本作为首要考虑因素的数据存储场景。
Redis社区版	无	兼容开源Redis，高性能。	适合中小型或验证型应用，适用于标准化Redis使用和迁移场景。

> **说明** 关于产品选型的详细介绍，请参见阿里云Redis产品选型必读。

特性对比

下述表格中，✔☺表示支持该功能，□表示不支持该功能。

> **说明** 混合存储型已停止售卖，更多信息，请参见【通知】Redis混合存储型实例停止售卖。推荐选择性能更强，功能更丰富的性能增强型实例。
> 若您已购买混合存储型，您可以通过提交工单迁移数据库实例。

类别	对比项	Redis企业版				Redis社区版	
		性能增强型	持久内存型	容量存储型	混合存储型（已停售）	2.8、4.0及5.0版本	6.0版本
基本性能	性能基准（以Redis社区版为基准）	300%	90%	读：50% 写：30%	90%~40% ②	一致	120%
	单个数据节点的最大连接数	30,000	10,000	10,000	10,000	10,000	10,000
	单Key服务能力（QPS参考值）①	450,000	130,000	60,000~30,000 ②	120,000~60,000 ②	140,000	160,000
规格特性	磁盘类型	本地盘 ESSD云盘（PL1）	Intel傲腾™持久化内存	ESSD云盘（PL1）	本地盘	本地盘	ESSD云盘（PL1）
	线程模型	多IO+单Worker（Real Multi-IO）③	单IO+单Worker	多IO+多Worker（Real Multi-IO）	单IO+多Worker	单IO+单Worker	多IO+单Worker
	单位成本（以Redis社区版为基准）	117%	70%	15%~20%	30%	一致	一致
数据结构	基础数据结构及命令支持	不同形态支持的命令有所不同，详情请参见Tair命令限制。				部分命令不支持，详情请参见Redis社区版命令支持。	
	数据结构模块集成	✔☺	✔☺☺☺（部分）	□	□	□	□
	主从复制一致性	最终一致	最终一致	最终一致	最终一致	最终一致	最终一致

类别	对比项	Redis企业版				Redis社区版	
		性能增强型	持久内存型	容量存储型	混合存储型（已停售）	2.8、4.0及5.0版本	6.0版本
	落盘一致性 ④	Write Back	Write Through	Write Through	Write Back	Write Back	Write Back
	持久化级别	秒级	命令级	命令级	秒级	秒级	秒级
安全性	数据库审计	✔	✘	✘	✘	✔	✘
	SSL加密	✔	✔	✘	✔	✔	✘
	IP白名单	✔	✔	✔	✔	✔	✔
性能分析	实时Top Key统计	✔	✔	✔	✘	✔	✔
	查询历史热点Key	✔	✘	✘	✘	✔	✘
	查询实时大Key	✔	✔	✘	✘	✔（2.8版本不支持）	✔
	离线分析大Key	✔	✔	✔	✘	✔	✔
高级功能	通过数据闪回按时间点恢复数据	✔	✘	✘	✘	✘	✘
	代理查询缓存	✔	✘	✘	✘	✘	✘
	全球分布式缓存	✔	✘	✘	✘	✘	✘
	DTS单向同步	✔	✘	✘	✔	✔	✔
	DTS双向同步	✔	✔	✔	✘	✘	✘

表格中数字标记的解释如下：

- ①：该QPS（每秒访问次数）参考值以时间复杂度为O(1)的命令衡量，时间复杂度越高，QPS参考值会相应降低。
- ②：该性能与数据访问的冷热分布相关，命中内存的比例越高性能越接近社区版基准性能。
- ③：区别于Redis社区版6.0的多线程，性能增强型的Real Multi-IO能够将IO加速地更彻底，具备更高的抗连接冲击性，且可以线性地提升吞吐能力。
- ④：数据落盘方式主要有下述两种：
  - Write Through：数据写入成功，数据同步落盘后返回。
  - Write Back：数据写入成功即返回成功，数据异步刷盘。

相关文档

- 企业版（Tair）简介
- 规格查询导航
- 架构信息查询导航

2.4. 本地盘和云盘实例对比

在购买Redis实例时，您可以选择本地盘或云盘实例来满足不同业务场景的需求。本文列出二者的主要区别，为您提供选型参考。

特性对比

对比项	本地盘实例	云盘实例
架构	基于云数据库Redis传统管控架构。	基于云数据库Redis新一代管控架构。 **说明** 后续的产品将基于此架构演进，同时将会提供本地盘实例到云盘实例的迁移功能。
扩容能力	• 扩容耗时较长。 • 集群架构的实例扩容会有闪断。 • 集群架构实例的分片节点的扩展数固定，例如2分片、4片、8分片等。	• 扩容能力比本地盘更好。 • 集群架构的实例扩容无闪断。 • 集群架构的实例支持自由调整分片节点的数量（最少1个分片节点），支持单分片的扩缩容，可更好地应对读写热点和倾斜。

云盘与本地盘功能支持度对比

当前云数据库Redis的本地盘实例功能较为完整，云盘版实例为云原生基础架构，集群架构的实例支持，未来会以云盘版为主要演进方向。详细的功能支持度对比如下表所示，为便于阅读，约定✔☺表示支持该功能，□表示不支持该功能，□表示无此概念。无感扩缩容

类别	功能	本地盘版			云盘版	
		标准架构	集群架构	读写分离架构	标准架构	集群架构
管理生命周期	变更实例配置	✔☺	✔☺	✔☺	✔☺ 支持无感扩缩容	✔☺ 支持无感扩缩容 支持自定义分片数（不受固定规格限制）
	重启实例	✔☺	✔☺	✔☺	✔☺	✔☺
	转包年包月	✔☺	✔☺	✔☺	□	□
	自动或手动续费	✔☺	✔☺	✔☺	✔☺	✔☺
	自定义分片数量	□	□	□	□	✔☺
	增、删分片数量	□	□	□	□	✔☺
	升级大版本	✔☺	✔☺	✔☺	□	□
	升级小版本	✔☺	✔☺	✔☺	✔☺	✔☺
	释放或退订实例	✔☺	✔☺	✔☺	✔☺	✔☺
	实例回收站	✔☺	✔☺	✔☺	✔☺	✔☺
管理网络连接	更换专有网络VPC或交换机	✔☺	✔☺	✔☺	□	□
	申请公网连接地址	✔☺	✔☺	✔☺	✔☺	✔☺
	集群架构支持直连访问模式	□	✔☺	□	□	✔☺
	修改连接地址或端口	✔☺	✔☺	✔☺	✔☺	✔☺
管理带宽	开启带宽弹性伸缩	✔☺	✔☺	✔☺	□	□
	手动调整实例带宽	✔☺	✔☺	✔☺	□	□

类别	功能	本地盘版			云盘版	
		标准架构	集群架构	读写分离架构	标准架构	集群架构
管理高可用	手动执行主备切换	✔	✔	✔	✘	✘
	重启或重搭代理节点	✘	✔	✔	✘	✘
	升级代理节点	✘	✔	✔	✘	✘
管理参数	设置实例参数	✔	✔	✔	✔	✔
管理标签	新建标签	✔	✔	✔	✔	✔
	根据标签筛选实例	✔	✔	✔	✔	✔
	解绑或删除标签	✔	✔	✔	✔	✔
其他管理功能	设置可维护时间段	✔	✔	✔	✔	✔
	更换实例所属的可用区	✔	✔	✔	✔	✘
	导出实例列表	✔	✔	✔	✔	✔
账号与安全	创建与管理账号	✔	✔	✔	✔	✔
	修改或重置密码	✔	✔	✔	✔	✔
	设置IP白名单	✔	✔	✔	✔	✔
	添加安全组	✔	✔	✔	✔	✘
	设置SSL加密	✘	✔	✘	✘	✘
	开启专有网络免密访问	✔	✔	✔	✔	✔
	开启实例释放保护	✔	✔	✔	✘	✘
性能与监控	查看监控数据	✔	✔	✔	✔	✔
	自定义监控项（旧版）	✔	✔	✔	✔	✔
	修改监控频率（旧版）	✔	✔	✔	✔	✔
	报警设置	✔	✔	✔	✔	✔
	性能趋势	✔	✔	✔	✔	✔
	实时性能	✔	✔	✔	✔	✔
	实例会话	✔	✔	✔	✔	✔
	慢请求	✔	✔	✔	✔	✔
	延时洞察	✔	✔	✔	✔	✔
	离线全量Key分析	✔	✔	✔	✔	✔
	实时Top Key统计	✔	✔	✔	✔	✔
	发起实例诊断	✔	✔	✔	✔	✔
	开通新版审计日志	✔	✔	✔	✘	✘
	查询慢日志	✔	✔	✔	✔	✔

巢翀与日志	功能	本地盘版			云盘版	
		标准架构	集群架构	读写分离架构	标准架构	集群架构
	查询运行日志	✔	✔	✔	✔	✔
备份与恢复	自动或手动备份	✔	✔	✔	✔	✔仅支持自动备份
	下载备份文件	✔	✔	✔	✔	✔
	通过数据闪回按时间点恢复数据	✔	✔	□	□	□
	从备份集恢复至新实例	✔	✔	✔	✔	✔
扩展功能	全球多活	✔	✔	✔	□	□
	代理查询缓存	□	✔	□	□	□
	Tair扩展数据结构概览	✔	✔	✔	✔	✔
	多线程	✔	✔	✔	✔	✔

产品支持度

实例及其创建方法	支持的实例类型	支持的引擎版本	支持的架构
本地盘实例 创建Redis本地盘实例社区版或性能增强型实例	Redis社区版	4.0 5.0	集群架构 标准架构 读写分离架构
	企业版（性能增强型）	5.0	集群架构 标准架构 读写分离架构
云盘实例 创建Redis社区版云盘实例	Redis社区版	7.0 6.0 5.0	标准架构 集群架构
	企业版（性能增强型）	5.0	标准架构 集群架构
	企业版（持久内存型）	兼容Redis 6.0	标准架构 集群架构
	企业版（容量存储型）	兼容Redis 4.0	标准架构

2.5. 功能特性

云数据库Redis版支持多种架构，数据可持久化存储，可用性高，且支持弹性扩展和智能运维。

架构灵活

- 双机热备架构
  系统工作时主节点（Master）和备节点（Replica）数据实时同步，主节点故障时系统自动进行秒级切换，备节点接管业务（期间会有秒级的闪断），主备架构保障系统服务具有高可用性。详情请参见标准版-双副本。
- 集群架构
  集群（Cluster）实例采用分布式架构，每个节点都采用一主一从的高可用架构，能够进行自动容灾切换和故障迁移。多种集群规格可适配不同的业务压力，可按需扩展数据库性能。详情请参见集群版-双副本。
- 读写分离架构

读写分离（Read-Write Splitting）实例由Proxy服务器、主备节点及只读节点组成提供高可用、高性能、高灵活的读写分离服务，解决热点数据集中及高并发读取的业务需求，最大化地节约用户运维成本。详情请参见读写分离版。

数据安全

- 数据持久化存储
  采用内存加硬盘的混合存储方式，在提供高速数据读写能力的同时满足数据持久化需求。
- 备份及一键恢复
  每天自动备份数据，数据容灾能力强，免费支持数据一键恢复，有效防范数据误操作，极大地降低发生业务损失的可能性。
- 多层网络安全防护
  - VPC私有网络在TCP层直接进行网络隔离保护。
  - DDoS防护实时监测并清除大流量攻击。
  - 支持配置1000个以上的IP白名单地址，从访问源进行风险控制。
  - 支持密码访问鉴权方式，确保访问安全可靠。
- 深度内核优化
  阿里云专家团队对源码Redis进行深度内核优化，有效防止内存溢出，修复安全漏洞，为您保驾护航。

高可用性

- 主从双节点
  标准版与集群版的双副本实例均有主从双节点，避免单点故障引起的服务中断。
- 自动检测与恢复
  自动侦测硬件故障，发生故障时能够进行故障转移，在数秒内恢复服务。
- 资源隔离
  实例级别的资源隔离可以更好地保障单个用户服务的稳定性。

弹性扩展

- 数据容量扩展
  云数据库Redis版支持多种内存规格的产品配置，您可以根据业务量升级内存规格。
- 性能扩展
  支持集群架构下弹性扩展数据库系统的存储空间及吞吐性能，突破海量数据高QPS性能瓶颈，轻松应对每秒百万次的读写需求。

相关操作方法请参见变更实例配置。

智能运维

- 监控平台
  提供CPU使用率、连接数、磁盘空间利用率等实例信息实时监控及报警，随时随地了解实例动态，更多信息请参见可观测性能力介绍。
- 可视化管理平台
  管理控制平台对实例克隆、备份、数据恢复等高频高危操作可便捷地进行一键式操作。
- 可视化DMS平台
  专业的DMS数据管理平台，提供可视化的数据管理，全面提升研发、运维效率。
- 数据库内核版本管理
  主动升级，快速修复缺陷，免去日常版本管理苦恼；优化Redis参数配置，最大化利用系统资源。

2.6. 应用场景

云数据库Redis版适合多种场景中的数据存储，尤其是请求并发量大的场景。

游戏行业应用

游戏行业可以选择云数据库Redis版作为重要的部署架构组件。

- 场景一：Redis作为存储数据库使用
  游戏部署架构相对简单，主程序部署在ECS上，所有业务数据存储在Redis中，作为持久化数据库。云数据库Redis版支持持久化功能，主备双机冗余数据存储。
- 场景二：Redis作为缓存加速应用访问
  Redis作为缓存层，加速应用访问。数据存储在后端的数据库中（RDS）。

Redis的服务可靠性至关重要，一旦Redis服务不可用，将导致后端数据库无法承载业务访问压力。云数据库Redis版提供双机热备的高可用架构，保障极高的服务可靠性。主节点对外提供服务，当主节点出现故障，系统自动切换备用节点接管服务，整个切换过程对用户全部透明。

电商行业应用

电商行业中对于Redis大量使用，多数在商品展示、购物推荐等模块。

- 场景一：秒杀类购物系统
  大型促销秒杀系统，系统整体访问压力非常大，一般的数据库根本无法承载这样的读取压力。云数据库Redis版支持持久化功能，可以直接选择Redis作为数据库系统使用。
- 场景二：带有计数系统的库存系统
  使用RDS存储商品的所有信息，通过云数据库Redis来存储应商品的库存信息。在高并发、大流量场景下，Redis的极高性能可以轻松应对每一次来自用户的商品库存查询，避免所有查询请求全部进入RDS，导致业务响应变慢等问题，提升用户体验。

视频直播类应用

视频直播类业务往往会重度依赖Redis业务去存储用户数据及好友互动关系。

- 双机热备保障高可用
  云数据库Redis版提供双机热备的方式，可以极大的提高服务可用性。
- 集群版解决性能瓶颈
  云数据库Redis版提供集群版实例，破除Redis单线程机制的性能瓶颈，可以有效的应对视频直播类流量突起，有效地支撑高性能的需求。
- 轻松扩容应对业务高峰
  云数据库Redis版可支持一键扩容，整个升级过程对用户全透明，可以从容应对流量突发对业务产生的影响。

2.7. 灾备方案介绍

云数据库Redis作为高性能的Key-Value数据库，在业务场景中往往承载着大量的重要数据，为保障数据安全性，云数据库Redis提供了多种灾备方案供您选择。

云数据库Redis容灾架构演进

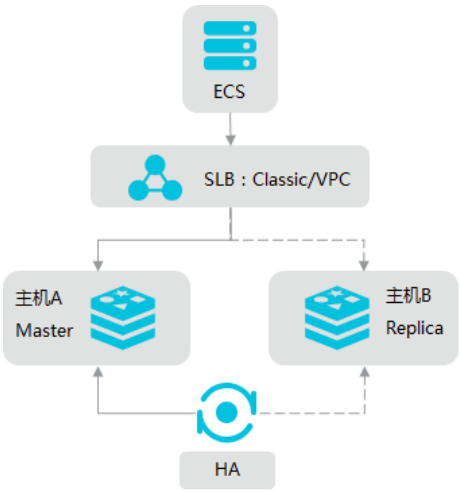
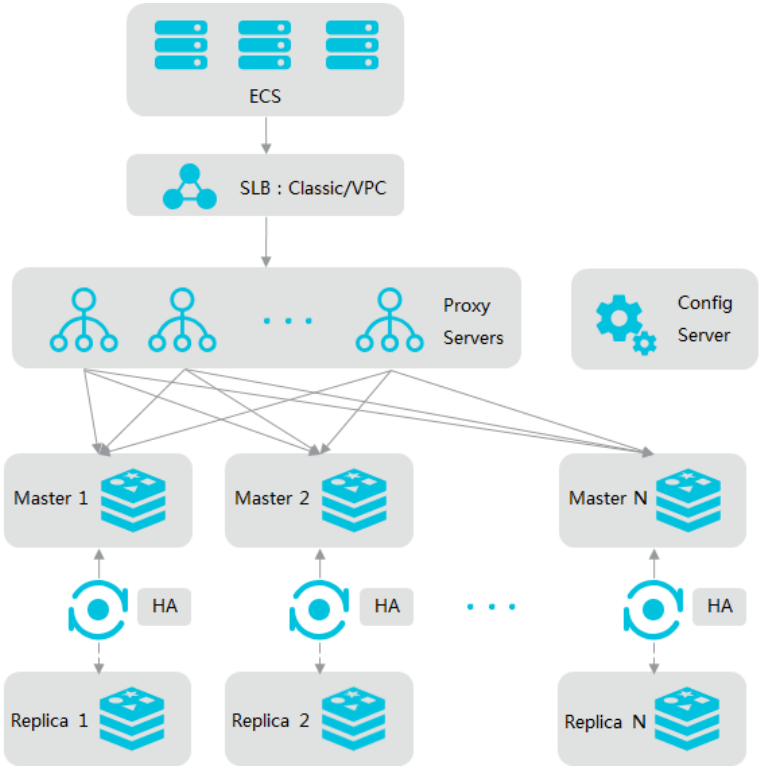
当云数据库Redis实例因不可预料的原因（例如设备故障、机房断电等）发生故障，容灾机制可用于保障数据的一致性和业务可用性。云数据库Redis提供多种灾备方案供您选择，可满足不同的业务场景。

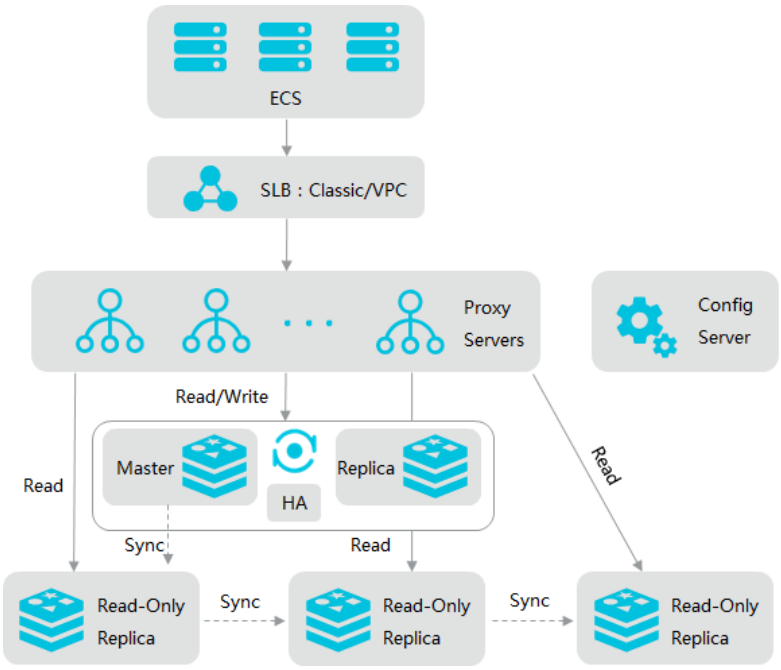
Redis容灾架构演进

灾备方案	灾备级别	说明
单可用区高可用方案	★★★☆☆	主备节点部署在同一可用区中的不同机器上，当任一节点发生故障时，由高可用HA（High Availability）系统自动执行故障切换，避免单点故障引起的服务中断。
同城容灾方案	★★★★☆	主备节点分别部署在同一地域下两个不同的可用区，当任一可用区因电力、网络等不可抗因素失去通信时，高可用HA系统将执行故障切换，确保整个实例的持续可用。
跨地域容灾方案	★★★★★	由多个子实例构成全球分布式实例，所有子实例通过同步通道保持实时数据同步，由通道管理器负责子实例的健康状态监测、主备切换等等异常事件的处理，适用于异地灾备、异地多活、应用就近访问、分摊负载等场景。

单可用区高可用方案

Redis全架构均支持单机房高可用架构。由高可用HA（High Availability）系统监控主备节点的健康状态并自动执行故障切换，避免单点故障引起的服务中断。

部署架构	说明
标准架构（双副本）	标准版-双副本高可用架构  标准架构（双副本）实例采用双机主从（Master-Replica）架构，高可用HA模块侦测到主节点故障时，会自动进行主从切换，将Replica提升为Master，而原来的Master恢复连接后会成为新的Replica。
集群架构（双副本）	集群版-双副本高可用架构  集群架构（双副本）实例中的数据分片用于承载数据，每个数据分片均为双副本（分别部署在不同机器上）高可用架构，主节点发生故障后，系统会自动进行主备切换保证服务高可用。关于各组件的详细介绍，请参见集群架构。

部署架构	说明
读写分离架构	读写分离版高可用架构  • 自动监控各节点的健康状态，异常时发起主备切换或重搭只读节点，并更新相应的路由及权重信息。 • Proxy会实时探测只读节点的状态，当出现下述情况时，Proxy会执行流量管控动作： ○ 只读节点处于异常状态：Proxy会降低该节点的服务权重，如果多次无法连接该节点，Proxy会停止该节点的服务（即不再将流量转发至该节点），待该异常被修复后重新启用该节点。 ○ 只读节点处于全量同步状态：Proxy会暂时停止该节点的服务，直到该节点完成全量同步。 关于各组件的详细介绍，请参见读写分离架构。

同城容灾方案

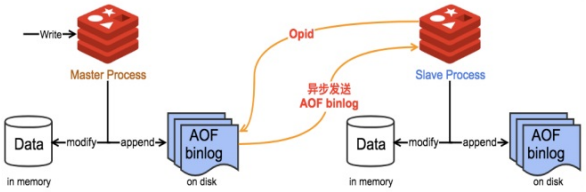
Redis标准版和集群版提供跨双机房的同城容灾架构。如果业务为单一地域部署，且对容灾要求较高，可在创建云数据库Redis实例时，选择支持同城容灾的可用区（即多可用区）。操作方法，请参见创建实例。

创建同城容灾实例

完成创建后，备机房将创建与主机房相同规格的Replica实例，主备机房的实例数据通过专门的复制通道同步。

当主机房出现电力或网络问题时，Replica实例将升级为Master实例，系统调用Config Server接口为Proxy更新路由信息。同时，云数据库Redis优化了Redis的同步机制，在同步位点上借鉴MySQL的GTID，实现了全局Opid，查找Opid的操作通过后台线程无锁进行，发送AOF binlog是异步同步的过程（可限流），保障了Redis服务的性能。

同城容灾实例的数据同步过程



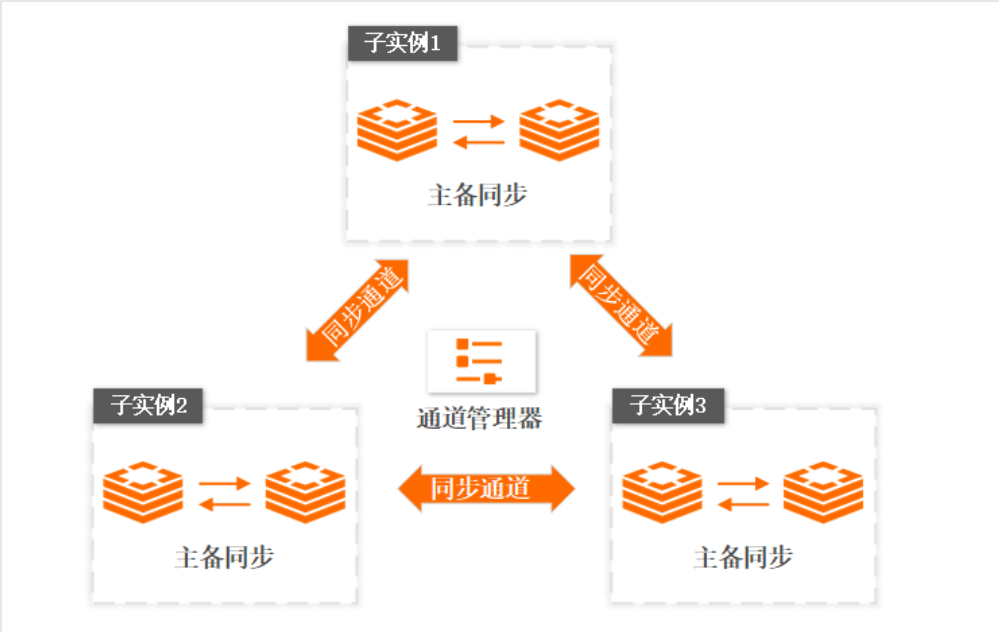
跨地域容灾方案

随着业务的快速发展，在业务分布较广时，如果还采用跨地域远距离访问的架构，将导致访问的延迟大，影响用户体验。借助阿里云的Redis全球分布式缓存功能，可帮助您解决业务因跨地域访问导致延迟大的问题，分布式缓存功能具有如下优势：

- 可直接创建或指定需要同步的子实例，无需通过业务自身的冗余设计来实现，极大降低业务设计的复杂度，让您专注于上层业务的开发。
- 可提供跨域复制（Geo-replication）能力，快速实现数据异地灾备和多活。

该功能可应用于跨地域数据同步场景及多媒体、游戏、电商等行业的全球化业务部署等场景。更多介绍，请参见Redis全球多活简介。

Redis全球分布式缓存架构



2.8. 可观测性能力介绍

相比原生Redis，云数据库Redis版提供了维度更广、种类更多及功能更强大的可观测性能力（Observability）。

背景信息

可观测性是以系统的指标、日志、链路追踪三大数据支柱为基础，衍生出如数据监控、问题分析、系统诊断等一系列的能力。

- 指标（Metrics）：记录一段时间内各个维度的量化信息，用来观察系统的某些状态和趋势。
- 日志（Logs）：记录程序运行过程中产生的一些离散事件。
- 链路追踪（Traces）：记录一次请求从接收到处理完成整个生命周期内的调用链路。

同时，云数据库Redis版还基于三大数据支柱进行信息聚合，提供数据分析能力，下表为云数据库Redis与原生Redis的可观测性能力对比。为便于浏览和内容表达，表格约定使用下述注释：

- ✔表示支持。
- ✘表示不支持。

- □表示不涉及。

可观测性能力		原生Redis	云数据库Redis	Tair（Redis企业版）
指标	性能指标	✔☺	✔☺（更细化）	✔☺（更细化）
日志	运行日志	✔☺	✔☺	✔☺
	慢日志	✔☺	✔☺	✔☺
	审计日志	□	✔☺	✔☺
	延时洞察	□	✔☺	✔☺
链路追踪	□	□	□	□
分析能力	实时热Key分析	□	✔☺	✔☺
	实时大Key分析	□	✔☺	✔☺
	离线全量Key分析	□	✔☺	✔☺
	实例诊断	□	✔☺	✔☺

指标

原生Redis提供了丰富的统计指标，包含Memory（内存分配、内存使用、内存碎片率情况等）， Stats（连接数、命令、网络、同步状态等）、CPU使用情况、Keyspace信息等。云数据库Redis版结合用户的使用体验，在原生Redis的基础上增加了更细化的指标，例如读QPS、写QPS等，更多信息请参见查看性能指标。

与此同时，云数据库Redis版的指标可观测性能力还具备如下优势：

- 实时性能：实时展示指标信息。
- 实例会话：实时展示Redis实例与客户端间的会话信息。
- 性能趋势：支持绘制任意时间跨度的曲线。

日志

云数据库Redis版提供了查询运行日志、慢日志、审计日志、延时洞察等功能。

- 运行日志（Redis log）
  按行输出Redis运行过程中的日志信息，记录运行过程中持久化、同步复制、报错信息以及代码中定义的调试记录等。
  在控制台目标实例详情页的**日志管理 > 运行日志**页签中，查看该实例的运行日志信息，更多信息请参见查询运行日志。
- 慢日志（Slowlog）
  记录Redis中执行时间（不含命令排队与网络传输时间）超过指定阈值的请求，慢日志信息包含执行时间戳、执行时长、命令参数、客户端信息等。您可以通过该功能第一时间查询耗时过长的命令列表，并进行相应优化，避免线上服务发生阻塞。
  在控制台目标实例详情页的**日志管理 > 慢日志**页签中，查看该实例的慢日志信息，更多信息请参见查询慢日志。
- 审计日志（Audit log）
  云数据库Redis版基于日志服务SLS（Log Service），提供审计日志功能，每条审计日志包含日志类型、执行时长、DB序号、客户端IP、账户名、命令详细信息以及扩展信息等。基于该功能，为您提供在线查询、分析操作日志（包含敏感操作`FLUSHALL`、`FLUSHDB`、`DEL`等）、慢日志及运行日志等，并且支持导出。
  在控制台目标实例详情页的**日志管理 > 审计日志**页签中，查看该实例的审计日志信息，更多信息请参见审计日志。
- 延时洞察（Latency metric）
  延时洞察是云数据库Redis提供的升级版延时统计功能，支持记录多达27个事件及所有Redis命令的执行耗时，并支持保存最近3天内所有的延时统计数据。
  在控制台目标实例详情页的**CloudDBA > 延时洞察**页签中，查看该实例的时延信息，更多信息请参见延时洞察。

分析能力

分析能力是基于指标、日志、链路追踪三大基础数据进行的信息聚合，是云数据库Redis版重要的服务能力。

- 热Key与大Key分析
  当某个Key接收的访问次数显著高于其它Key时，可以将其称为热Key（Hotkeys），若未能及时处理热Key可能会导致访问倾斜甚至缓存击穿等问题；当某个Key含有较多数据成员或者占用较大内存时，可以将其称为大Key（Big keys），若未能及时处理大Key会导致执行命令的耗时增加，严重时甚至引发内存溢出（Out Of Memory）。

您可以通过云数据库Redis版的**实时Top Key统计**功能，帮助定位热Key与大Key，**实时Top Key统计**功能支持实时展示实例中的热Key和大Key信息，同时支持查看4天内大Key和热Key的历史信息。**实时Top Key统计**功能准确性高，且对性能几乎无影响，帮助您掌握Key在内存中的占用、Key的访问频次等信息，溯源分析问题，为您的优化操作提供数据支持。
在控制台目标实例详情页的**CloudDBA > 实时Top Key统计**页签中，进行热Key与大Key分析，更多信息请参见实时Top Key统计。

- 离线全量Key分析
  离线全量Key分析功能支持全数据结构、全实例架构及Redis各个版本的离线RDB备份文件解析，对线上服务无影响。相比开源工具redis-rdb-tool的解析速度，**离线全量Key分析**在大小Key混合（占比1：9）的场景下实现4倍速度提升，在中大Key场景下实现20倍速度提升，同时保证进程内存占用固定维持在1 GB以内，避免大Key解析可能带来内存溢出的问题。**离线全量Key分析**还提供了最长子元素查询，方便进一步业务排查。
  在控制台目标实例详情页的**CloudDBA > 离线全量Key分析**页签中进行分析，更多信息请参见离线全量Key分析。
- 实例诊断
  云数据库Redis版综合了性能指标、慢日志、key分析等能力，提供了一站式全链路的实例诊断功能，从性能水位、访问倾斜情况、慢日志等多方面评估实例的健康状况，并给出改善建议，极大程度地提高了Redis实例的自动化运维能力，降低使用成本。
  在控制台目标实例详情页的**CloudDBA > 诊断报告**页签中，进行实例诊断，更多信息请参见实例诊断。

3.云数据库Redis企业版（Tair）

3.1. 企业版（Tair）简介

阿里云数据库Redis企业版（又称阿里云Tair），是基于阿里集团内部使用的Tair产品研发的云上托管企业级内存数据库，从2009年开始正式承载阿里集团业务，历经天猫双十一、优酷春晚、菜鸟、高德等业务场景的磨练，是一款真正的企业级内存数据库产品。

阿里云Tair的诞生

2004年，淘宝开始应用缓存技术。最先投入应用的是基于前端页面的缓存技术，采用ESI来标识可以加速和不能加速的网页内容片段，有效减少了从服务端抓取整个页面的次数。

随着淘宝网的流量快速增长，数据库的压力与日俱增，基于后端系统的缓存技术应运而生。从服务淘宝详情和验证码等业务的持久化系统TBStore，到初始服务于淘宝用户中心的TDBM等等，后端系统缓存技术经历了多个系统和阶段的演变与积累，到2009年，这些系统、技术经验经过进一步的研发，融合成了阿里巴巴大规模高性能内存数据库Tair。

如今，基于Tair演进的Redis企业版已经是阿里巴巴集团调用量最大的系统之一，在多年的阿里巴巴双十一全球狂欢节上提供了核心的在线访问加速能力，承受住了每秒数亿次的调用。

阿里云Tair的发展

时间	事件
2021年07月	发布新产品系列： • 持久内存型：基于Intel 傲腾™傲腾持久内存，为您提供大容量、兼容Redis的内存数据库产品。单实例成本对比Redis社区版最高可降低30%，且数据持久化不依赖传统磁盘，保证每个操作持久化的同时提供近乎Redis社区版的吞吐和延时，极大提升业务数据可靠性。 • 容量存储型：基于云盘ESSD研发，兼容Redis核心数据结构与接口，可提供大容量、低成本、强持久化的数据库服务。 未来，Tair将重点建设云原生，如软硬件技术结合、数据智能分布、数据存储和计算处理一体化等核心能力。
2019年11月	发布Tair 3.0，即云数据库Redis企业版： • 性能增强型：采用多线程模型，100%兼容原生Redis，集成多个自研Tair数据结构，提供高性能、高兼容性及带有诸多企业级特性的数据库服务。
2019年04月	KVStore团队在Redis开源社区贡献排名前三，并在RedisConf 2019上发表了公开演讲。
2018年08月	KVStore在中国率先推出混合存储实例，冷热数据分离，有效降低大客户使用成本。
2017年11月	Tair热点散列经过双十一考验，解决了业内的缓存热点难题。
2017年04月	Tair 2.0上线，开始支持高德、优酷新BU。 云上OCS全面升级为KVStore。
2016年08月	Tair智能运维平台上线，助力2016双十一迈入千亿时代。
2015年03月	Tair推出阿里云KVStore，即云数据库Redis版，真正进入了云时代。
2014年05月	Tair推出阿里云上缓存产品OCS，成为阿里云初始的基础产品之一，服务云上Memcache用户。
2013年04月	Fastdump系统落地，大幅度降低导入时间和访问延时。 Tair在阿里妈妈获得规模化应用。
2012年10月	推出RDB缓存引擎，引入类Redis接口，支持更灵活、复杂的数据结构。
2011年06月	上线LDB持久化引擎，满足互联网KV存储需求。
2009年11月	Tair的第一个双十一，正式开始支撑超大流量场景。
2009年04月	Tair 1.0正式诞生，并被应用于淘宝核心系统、MDB缓存、用户中心等业务。

产品类型及特性

随着互联网的高速发展，业务场景变得越来越丰富和复杂，Redis企业版作为一个高可用、高性能的分布式NoSQL数据库，从访问延时、持久化需求、整体成本这三个核心维度考量，基于DRAM、NVM和ESSD云盘存储介质，推出了多种不同形态的产品，为您提供更强的性能、更多的数据结构和更灵活的存储方式，满足不同场景下的业务需求。

Redis企业版产品类型	特性
性能增强型	• 采用多线程模型：性能约为同规格社区版实例的3倍。 • 支持多种增强型数据结构模块（modules），包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，使业务无需再关心存储的结构和时效性，能够极大提升业务开发效率。 • 超高兼容性：100%兼容原生Redis，无需修改业务代码。 • 支持诸多企业级特性：通过数据闪回按时间点恢复数据、代理查询缓存、全球多活等。
持久内存型	• 超高性价比：相同容量下对比Redis社区版，价格降低30%左右，性能可达原生Redis的90%。 • 支持增强型数据结构模块（modules）：TairString（含CAS和CAD命令）、TairCpc。 • 掉电数据不丢失：强大的命令级持久化保障，每个写操作持久化成功后返回，可将其作为内存数据库（非缓存）使用。 • 大规格优化：解决大规格下执行AOF重写调用fork引起的延时抖动等问题。 • 高兼容性：兼容绝大部分原生Redis的数据结构和命令。
容量存储型	容量存储型基于云盘ESSD研发，兼容Redis核心数据结构与接口，可提供大容量、低成本、强持久化的数据库服务。容量存储型在降低成本和提升数据可靠性的同时，也解决了原生Redis固有的因fork而预留部分内存的问题。适用于兼容Redis、需要大容量且较高访问性能的温冷数据存储场景。
混合存储型（停止售卖）	混合存储型采用内存加磁盘的存储模式，能够在业务高峰期后对冷热数据进行弹性分离，既保障了热数据的内存访问速度，又提供了远超社区Redis的存储容量，实现了性能与成本的平衡。 **说明** 混合存储型已停止售卖，更多信息，请参见【通知】Redis混合存储型实例停止售卖。推荐选择性能更强，功能更丰富的性能增强型实例。 若您已购买混合存储型，您可以通过提交工单迁移数据库实例。

购买指引

说明 关于下表中本地盘和云盘版的区别，请参见本地盘和云盘实例对比。

Redis企业版产品类型	购买链接	操作说明
性能增强型	立即购买	创建Redis本地盘实例社区版或性能增强型实例
	立即购买	创建Redis企业版云盘实例持久内存型或容量存储型实例
持久内存型	立即购买	创建Redis企业版云盘实例持久内存型或容量存储型实例
容量存储型		

相关文档

- 性能增强型性能测试
- 持久内存型性能测试
- 容量存储型性能测试

3.2. 性能增强型

Redis企业版性能增强型（简称性能增强型）适合并发量大、读写热点多，对性能要求超过云Redis社区版实例的场景。相比云数据库Redis社区版，性能增强型主要在多线程性能增强和多模块集成方面进行了优化。

主要优势

类别	说明
兼容性	• 100%兼容原生Redis，无需修改业务代码。
性能	• 采用多线程模型，性能约为同规格社区版实例的3倍，能够突破热点数据高频读写受到的性能限制。 • 相比原生Redis，高QPS场景下响应时间更低，性能表现更佳。 • 在大并发场景下运行稳定，可以极大地缓解突发大量请求导致的连接问题，从容应对业务高峰。 • 全量同步和增量同步在IO线程中进行，提高同步速度。
部署架构	• 支持标准、集群和读写分离部署架构。
数据结构模块集成	• 集成多个自研的Redis模块，包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，扩展了Redis的适用性，使业务无需再关心存储的结构和时效性，能够极大提升业务开发效率。
企业级特性	• 支持诸多企业级特性：通过数据闪回按时间点恢复数据、代理查询缓存、全球多活等。
数据安全	• 支持开启SSL加密，提升通信数据的安全性。 • 支持开启透明数据加密TDE，对RDB数据文件执行加密和解密，提升数据安全性。

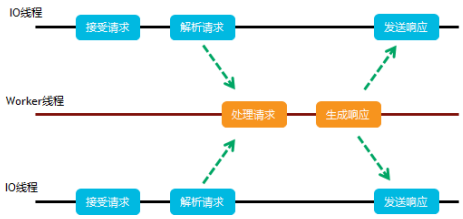
适用场景

适用于视频直播、电商秒杀和在线教育等场景，下面列举了性能增强型在4个典型场景中的应用。

- 场景1：使用Redis社区版的标准版-双副本实例在秒杀场景中构建缓存，部分热点Key的QPS要求高达20万以上，社区版实例无法满足业务高峰期的需求。
  采用性能增强型（标准架构）实例后，热门商品秒杀过程流畅，未发生性能问题。
- 场景2：在业务中使用云Redis社区版集群实例，但在使用事务和Lua脚本功能时有一定的限制。
  采用性能增强型实例后，在满足性能需求的同时消除了集群版的命令使用限制。
- 场景3：自建有一主多备的Redis服务，随着业务中访问量的不断提高，备节点数量也要随之增加，管理维护成本越来越高。
  采用具备一个数据节点五个只读副本的性能增强型（读写分离架构）实例后，可以轻松应对百万级QPS的业务挑战。
- 场景4：自建有Redis集群来承担线上千万级QPS的业务压力。随着业务的发展，Redis分片数不断增加，管理维护成本居高不下。
  采用性能增强型（集群架构）实例后，集群规模缩到原来的三分之一，管理维护成本大幅降低。

线程模型对比

线程架构	说明
Redis单线程模型 	Redis社区版和原生Redis采用单线程模型，数据处理流程为：读取请求，解析请求，处理数据，发送响应。其中网络IO和请求解析占用了大部分的资源。

线程架构	说明
Redis多线程模型 	性能增强型将Redis服务各阶段的任务进行分离，通过分工明确的多个线程并行处理各阶段任务，达到提高性能的目的。 • IO线程：负责请求读取、响应发送、命令解析等。 • Worker线程：负责命令处理、定时器事件等。 • 辅助线程：负责高可用探测、保活等。 IO线程读取用户的请求并进行解析，之后将解析结果以命令的形式放在队列中发送给Worker线程处理。Worker线程将命令处理完成后生成响应，通过另一条队列发送给IO线程。 Redis性能增强型最多支持4个IO线程并发运行。为了提高线程的并行度，IO线程和Worker线程之间采用无锁队列和管道进行数据交换。 **说明** • 对于通用的类型，例如：String、List、Hash、Set、Zset、 Hyperloglog、Geo，以及扩展结构都有很好的加速效果。 • 对于pub、sub、blocking系列API，由于其复制是在worker中完成的，可被加速提升吞吐，性能提升约50%。 • 由于transaction、lua script要求串行执行，无加速效果。

说明 区别于Redis社区版6.0的多线程（性能至多提升2倍，且CPU资源消耗高），性能增强型的Real Multi-IO能够将IO加速地更彻底，具备更高的抗连接冲击性，且可以线性地提升吞吐能力。

性能对比

云Redis社区版实例采用与原生Redis相同的单线程模型，每个数据节点支持8万到10万的QPS；Redis性能增强型采用多线程模型，由IO线程、Worker线程和辅助线程共同完成数据处理，单节点性能为社区版实例的3倍左右。下表展示了不同架构下，社区版和企业版（性能增强型）实例的适用场景对比。

架构	实例类型	说明
标准架构	社区版	不适用于单节点QPS要求超过10万的场景。
	企业版（性能增强型）	可应用于QPS高于10万的场景。
集群架构	社区版	包含多个数据节点，每个节点的性能与标准版实例相似。当某个节点储存了热度较高的数据并面临大并发量的请求时，该节点中其它数据的读写可能受到影响，形成性能瓶颈。
	企业版（性能增强型）	能更好地应对热读写，降低维护成本。
读写分离架构	社区版	有较高的读性能，在读多写少的场景表现良好，但不适用于大并发写入的场景。
	企业版（性能增强型）	既有较高的读性能，又能承受大并发写入，适用于写请求多而读请求更多的场景。

数据结构模块集成

云Redis社区版与开源Redis相同，支持String、List、Hash、Set、Sorted Set、Stream等数据类型，能够满足大部分场景下的开发需求，但在一些复杂场景中，原生数据类型无法直接满足某些业务需求，只能使用通过调整应用数据、使用Lua脚本等方式来实现。

性能增强型集成多个自研的Redis模块，包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，扩展了Redis的适用性，同时降低了复杂场景下业务的开发难度，让您专注于业务创新。

扩展数据结构	说明
CAS和CAD命令	为增强Redis String的功能而开发的命令，可以实现简洁高效的Redis分布式锁。
TairString	TairString是一种带版本号的String类型数据结构，TairString除包含Key和Value外，还携带了版本（version）信息。除此之外，TairString在Redis String加减功能的基础上支持了边界设置，可以将INCRBY、INCRBYFLOAT的结果限制在一定的范围内，超出范围则提示错误。
TairHash	TairHash是一种Hash类型的数据，不但和原有的Redis Hash一样支持丰富的数据接口和高处理性能，还支持为field设置过期时间和版本，简化业务开发流程。通过高效的Active Expire算法，可以在不对响应时间造成明显影响的前提下，更高效的完成对field的超时判断和删除。
TairGIS	TairGIS是一种使用R-Tree做索引，支持地理信息系统GIS（Geographic Information System）相关接口的数据结构。Redis的原生GEO命令使用1D索引，主要用于点的查询，TairGIS使用2D索引，支持点、线、面的查询，适合判断相交或包含关系，功能更加强大。
TairBloom	TairBloom是一种可动态扩容的布隆过滤器，完全兼容RedisBloom模块的命令，具有动态扩容的能力，相对传统实现方式消耗内存更低，可在扩容的同时维持误判率的稳定，适合需要高效判断大量数据是否存在且允许一定误判率的业务场景。
TairDoc	TairDoc是一种文档类型的数据结构，支持JSON标准，完全兼容ReJSON模块的命令，同时，TairDoc数据以二进制树的方式存储，支持对JSON中子元素的快速访问。
TairTS	TairTS是基于Redis Module开发的时序数据结构，提供低时延、高并发的内存读写访问，及快速的过滤、聚合查询功能，集存储与计算为一体，在简化了处理时序数据流程的同时，大幅度提高了性能。
TairCpc	TairCpc是基于CPC（Compressed Probability Counting）压缩算法开发的数据结构，支持仅占用很小的内存空间对采样数据进行高性能计算。
TairZset	TairZset可实现任意维度的double类型的分值排序，提升数据处理效率，且客户端适配简易，无需任何编解码封装，解决原生Sorted Set（也称Zset）只支持1个double类型的分值排序的局限性。
TairRoaring	TairRoaring是基于Tair引擎的Roaring Bitmap实现，通过2层索引和引入多种动态容器（Container），同时使用了包括SIMD instructions、Vectorization、PopCnt算法在内的等多种工程优化，提供更低的内存占用及更高的集合计算效率。
TairSearch	TairSearch是基于Redis module全自研（不基于Lucene等开源搜索库）的全文搜索模块，采用和Elasticsearch相似（ES-LIKE）的查询语法。

企业级特性

企业级特性	说明
通过数据闪回按时间点恢复数据	开启Redis的数据闪回功能后，Redis最长可将AOF备份数据保留7天，在此期间您随时可以指定一个精确到秒的时间点，系统会基于所选时间点的备份数据创建一个新的实例，实现精确的数据恢复。
代理查询缓存	开启代理查询缓存功能（Proxy Query Cache）后，代理节点会缓存热点Key对应的请求和返回信息，当在有效时间内收到同样的请求时直接返回结果至客户端，无需和后端的数据分片交互。更多信息，请参见通过Proxy Query Cache优化热点Key问题。
全球多活	Redis全球多活是基于云数据库Redis自研的多活数据库系统，可轻松实现异地多个站点同时对外提供服务的业务场景，助力企业快速复制阿里巴巴异地多活架构。
借助DTS实现双向数据同步	通过数据传输服务DTS（Data Transmission Service）实现Redis企业版实例间的双向数据同步，可应用于异地多活、数据容灾等多种场景，详情请参见Redis企业版实例间的双向同步。

常见问题

客户端不支持新模块的命令怎么办？

答：您可以先在应用代码中定义需要使用的新模块命令，然后再使用这些命令，或者直接使用阿里云在Jedis基础上开发的TairJedis客户端。

相关文档

- 性能增强型性能测试

- 步骤1：创建实例
- 规格查询导航

3.3. 持久内存型

Redis企业版持久内存型（简称持久内存型）基于Intel 傲腾™持久内存，为您提供大容量、兼容Redis的内存数据库产品。单实例成本对比Redis社区版最高可降低30%，且数据持久化不依赖传统磁盘，保证每个操作持久化的同时提供近乎Redis社区版的吞吐和延时，极大提升业务数据可靠性。

购买方式

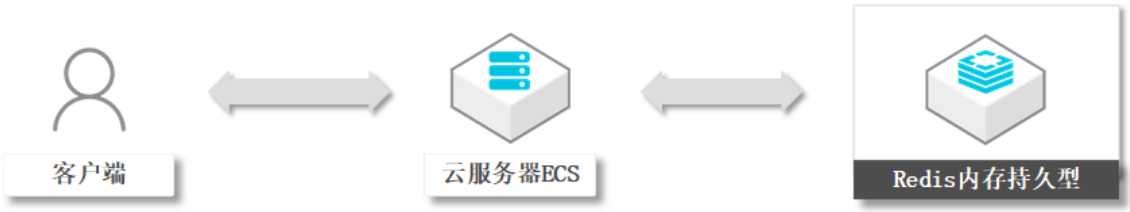
创建Redis企业版云盘实例持久内存型或容量存储型实例

背景信息

由于内存的价格相对昂贵且容量具备较大的局限性，限制了在某些场景中的规模化使用。阿里云于2018年正式开始投入Intel持久化内存的研究和落地，成功应用于当年双11的电商商品核心集群中，大幅降低了成本，是在生产环境正式部署应用Intel持久化内存硬件的产品。

随着云上环境的成熟和持久内存相关技术的完善，阿里云基于持久内存全新研发了数据持久落地的自研引擎，结合神龙裸金属服务器推出了Redis持久内存型产品，将传统Redis内存易失性演进到了持久内存原生持久化能力，大幅降低数据丢失的风险。

持久内存型产品既拥有内存级的访问延时和吞吐，也拥有数据持久化的能力。除了降低成本之外，持久内存型还能带来应用架构的简化，可将目前流行的应用+缓存+持久存储的架构模型，演进为更加简洁的应用+具备持久能力的内存数据库的架构模型，如下图所示。



产品优势

持久内存型基于Intel傲腾™持久化内存硬件，提供大容量、兼容Redis的内存数据库产品，数据持久化不依赖传统磁盘，保证每个操作持久化的同时提供近乎Redis社区版的吞吐和延时，极大提升业务数据可靠性。适用于兼容Redis、大容量、服务抖动稳定可控，数据持久化要求高的热温数据存储场景。

优势项	说明
超高性价比	• 相同容量下对比阿里云Redis社区版，价格降低30%左右。 • 性能可达到原生Redis的90%。
数据结构模块集成	• 支持TairString（含CAS和CAD命令）、TairCpc。
大规格优化	• 解决大规格下执行AOF重写调用fork引起的延时抖动、服务数据加载慢等问题，无需在性能与持久化中取舍。 • 可提供128 ~ 512 GB的大容量主从规格，集群版预计最大规格为32 TB。
掉电数据不丢失	• 强大的命令级持久化保障，每个写操作持久化成功之后返回。 • 相较于Redis的秒级数据丢失，在高QPS场景下数据更加有保障。
高兼容性	• 完全适配现有阿里云Redis数据库体系，具备高可用、弹性扩容缩容、日志、智能诊断与灵活的备份还原服务能力。 • 兼容原生Redis绝大部分的数据结构和接口。

适用场景

- 海量数据下对性能与成本要求高的场景
  计算中间数据对性能的要求很高，采用Redis社区版成本较高，如果采用HBase之类的数据库存储数据则可能无法满足性能需求。采用持久存储型实例保障数据持久化的同时提供近乎Redis社区版的吞吐和延时，可很好地平衡性能与成本。

- 最终数据存储持久化要求高的场景
  游戏场景直接采用持久存储型实例作为最终的数据存储，相较于使用Redis+MySQL的架构场景，可获得更简洁的架构，更高的性能和性价比，且数据更加可靠。

实例规格
持久内存型规格

相关文档
- Tair命令限制
- 持久内存型性能测试

3.4. 容量存储型
Redis企业版容量存储型（简称容量存储型）基于云盘ESSD研发，兼容Redis核心数据结构与接口，可提供大容量、低成本、强持久化的数据库服务。容量存储型在降低成本和提升数据可靠性的同时，也解决了原生Redis固有的fork问题而预留部分内存的问题。适用于兼容Redis、需要大容量且较高访问性能的温冷数据存储场景。

购买方式
创建Redis企业版云盘实例持久内存型或容量存储型实例

背景信息
传统Redis基于内存易失性存储介质，随着业务的持续快速发展，数据量的飞速增长，经常会遇到下述问题：

- 在大容量下的Redis存在服务抖动延时的问题。
- 在业务发展后期，沉淀的数据量越来越多，而访问量越来越低，此时性能存在过剩的情况。

为解决上述业务上遇到的问题，阿里云基于ESSD云盘存储介质，推出了容量存储型产品，成本最低可达到全内存版本的15%，拥有极高的性价比；容量可达到百TB级别，拥有更高的数据可靠性。容量存储型产品在降低成本和提升数据可靠性的同时，也解决了原生Redis固有的fork问题而预留部分内存的问题。

优势对比

对比项	容量存储型	混合存储型（已停售）（已停止售卖）	开源Pika
兼容性	兼容大部分原生Redis命令	完全兼容原生Redis命令	和原生Redis命令差别较大
持久化	可配置半同步或异步同步	异步同步	异步同步
数据分布	全磁盘（阿里云TairDB存储引擎）	内存中存放热数据的Key和Value，磁盘中存放所有的Key和Value	全磁盘（RocksDB存储引擎）
性能	约为原生Redis的60%，其中复杂数据结构（例如LIST、HASH、SET、ZSET）的性能比Pika高10%	内存中的热数据和原生Redis一致，冷数据性能不做保证，整体性能跟实际冷热数据负载相关	约为原生Redis的60%
适用场景	温数据	数据温度不确定	温数据
成本	最低为Redis社区版的15%	较容量存储型成本平均高20%	自建物理设备成本稍低，运维成本较大

实例规格
容量存储型规格

常见问题
- Q：容量存储型的引擎版本是什么？
  A：容量存储型采用的是阿里云自研的引擎版本（兼容4.0版本）。关于命令支持度的详细信息，请参见Tair命令限制。

> **说明** 为兼容某些组件或客户端对引擎版本的判断依赖（例如Redis-shake），在INFO命令的返回信息中保留了redis_version（值为4.9.9）。

相关文档

- 容量存储型性能测试

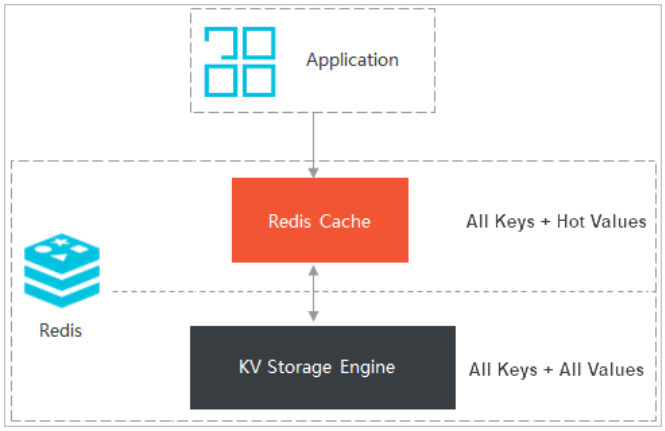
3.5. 混合存储型（已停售）

云数据库Redis企业版支持性能增强型和混合存储型。与社区版Redis不同，混合存储型整合了内存和磁盘二者的优势，在提供高速数据读写能力的同时满足了数据持久化的需求。

> **说明** 混合存储型已停止售卖，更多信息，请参见【通知】Redis混合存储型实例停止售卖。推荐选择性能更强，功能更丰富的性能增强型实例。
> 若您已购买混合存储型，您可以通过提交工单迁移数据库实例。

简介

混合存储型架构图



Redis企业版混合存储型（简称混合存储型）是阿里云自主研发的兼容Redis协议的混合存储产品，使用磁盘存储全量数据，将热数据保存到内存中供应用快速读写。在保证常用数据访问性能不下降的基础上，混合存储型能够大幅度降低用户成本，实现性能与成本的平衡，同时使单个Redis实例的数据量不再受内存大小的限制。

- 内存数据：内存中存放了热数据的Key和Value，同时为快速确认要操作的Key是否存在，内存中也会缓存所有的Key信息。
- 磁盘数据：磁盘中存放所有的Key和Value，Redis的数据结构（例如Hash）也会以一定的格式进行存储在磁盘。

适用场景

适用场景	说明
视频直播	视频直播类业务往往存在大量热点数据，大部分的请求都来自于热门的直播间。使用混合存储型，内存中保留热门直播间的数据，不活跃的直播间数据被自动存储到磁盘上，可以达到对有限内存的最佳利用效果。
电子商务	电商类应用往往有大量的商品数据。使用混合存储型可以轻松突破内存容量限制，将大量的商品数据都存储到混合存储型中。在正常业务请求中，活跃的商品数据会保留在内存，不活跃的商品数据会逐渐交换到磁盘上，从而解决内存不够的问题。
在线教育	在线教育类的场景有大量的课程、题库以及师生交流信息等数据，通常只有热门课程和最新题库会被频繁访问。使用混合存储型，将所有课程信息存储到磁盘，访问量大的课程和题库数据存储到内存并常驻内存，保证高频访问数据的读写性能，实现高性能与高性价比的有机结合。

典型业务场景的示例如下：

- 场景1：使用开源Redis集群存储了100GB的数据，但高峰期QPS不到2万，其中80%的数据的访问频率很低。
  使用32GB内存加128GB磁盘的混合存储型实例后，节省了近70GB的内存空间，存储成本下降50%以上。
- 场景2：在IDC自建Pika实例来解决Redis存储成本高的问题。总数据量约400GB，其中访问频率高的数据仅占10%左右，并且集群的运维成本居高不下。
  使用64GB内存加512GB磁盘的混合存储型实例后，既免除了繁重的运维工作，又保障了服务质量。

3.6. Tair（Redis企业版）扩展数据结构

3.6.1. Tair扩展数据结构概览

本文介绍云原生内存数据库Tair版集成的数据结构，以及与Redis Stack Server的功能对比。

云数据库Redis社区版与开源Redis相同，支持String、List、Hash、Set、Sorted Set、Stream等数据类型，能够满足大部分场景下的开发需求，但在一些复杂场景中，原生数据类型无法直接满足某些业务需求，需要通过开发大量代码、使用Lua脚本等复杂的方式实现。

Tair实例（云数据库Redis企业版）集成了多个自研的数据结构，包括TairString（包含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，从多方面扩展Redis的适用性，降低复杂场景下业务的开发难度，同时可以帮助您精简大量代码并提高业务整体性能，使您专注于业务创新。

> 说明
> - 性能增强型支持以上所有数据结构。
> - 持久内存型支持TairString（包含CAS和CAD命令）、TairCpc。

Tair扩展数据结构与Redis Stack模块（Modules）

类型	Tair扩展数据结构	Redis Stack Server	说明
String增强	• TairString • CAS和CAD命令	无	• TairString是一种带版本号的string类型数据结构，同时还在Redis String加减功能的基础上支持了边界设置，可以将INCRBY、INCRBYFLOAT的结果限制在一定的范围内，超出范围则提示错误。该数据结构已开源，更多信息请参见TairString。 • CAS和CAD命令可实现简洁高效的高性能分布式锁。 最佳实践：基于TairString实现高性能乐观锁、基于TairString实现高效限流器。
Hash增强	TairHash	无	TairHash支持为field设置过期时间和版本，提高了Hash数据结构的灵活性，简化了很多场景下的业务开发工作。该数据结构已开源，更多信息请参见TairHash。
Zset增强	TairZset	无	TairZset可实现256个维度的double类型的分值排序，提供和的能力。该数据结构已开源，更多信息请参见TairZset。普通排行榜分布式架构排行榜 最佳实践：基于TairZset轻松实现多维排行榜、基于TairZset实现分布式架构排行榜。
Doc（JSON）	TairDoc	RedisJSON	TairDoc完全兼容RedisJSON，同时支持JSONPointer和JSONPath两种语法，支持JSON格式到XML格式、YAML格式的转换。
Search	TairSearch	RedisSearch	TairSearch提供类似Elasticsearch（ES-LIKE）的语法，提供种类更多、效果更准确的分词器，查询性能更佳。
GeoSpatial	TairGIS	无	TairGIS是一种使用R-Tree做索引，支持地理信息系统GIS（Geographic Information System）相关接口的数据结构。支持点、线、面的查询，支持包含、被包含、相交等多种关系判断。 最佳实践：基于TairGIS轻松实现用户轨迹监测和电子围栏。
TimeSeries	TairTS	RedisTimeSeries	TairTS相比较RedisTimeSeries具备更强的标签（Tag）扩展能力，支持Skey（Tag）的两级Hash结构时间线，支持对Skey（Tag）进行二级时间线聚合查询，支持对历史时序数据的更新或累加等。 最佳实践：基于TairTS实现秒级监控。

类型	Tair扩展数据结构	Redis Stack Server	说明
Sketches	TairBloom	RedisBloom	TairBloom兼容RedisBloom，支持动态扩容，同时通过64位的Hash算法降低Hash碰撞率，显著降低大数据的冲撞率。 最佳实践：推荐系统、爬虫系统，更多信息请参见TairBloom。
	TairCpc	无	TairCpc是基于CPC（Compressed Probability Counting）压缩算法开发的数据结构，支持仅占用很小的内存空间对采样数据进行高性能计算，支持滚动窗口和滑动窗口，可以更好地支持流式运算，支持大数据分析中常用的聚合算子， 如：`DISTINCT`、`COUNT`、`MAX`、`MIN`、`FIRST`、`LAST`、`SQUARED`等。
Bitmap	TairRoaring	无	TairRoaring提供高效的计算模块和极高的稳定性，支持多位图运算能力，提升了性能和空间效率。 最佳实践：基于TairRoaring实现人群圈选方案。

常见问题

- Q：客户端不支持新模块的命令怎么办？
  A：您可以先在应用代码中定义需要使用的新模块命令，然后再使用这些命令。推荐直接使用阿里云在Jedis基础上开发的TairJedis客户端，通过TairJedis可直接调用Tair扩展数据结构。

3.6.2. TairString

本文介绍Redis企业版的TairString，是Tair团队自研，一种带版本号的String类型数据结构。

TairString简介

Redis的String仅由key和value组成，而TairString不仅包含key和value，还携带了版本（version），可用于乐观锁等场景。除此之外，TairString在Redis String加减功能的基础上支持了边界设置，可以将**INCRBY**、**INCRBYFLOAT**的结果限制在一定的范围内，超出范围则提示错误。

主要特性

- value携带版本号。
- 使用**INCRBY**、**INCRBYFLOAT**递增数据时可设置变更范围。

该Module已开源，更多信息请参见TairString。

最佳实践

- 基于TairString实现高性能分布式锁
- 基于TairString实现高性能乐观锁
- 基于TairString实现高效限流器

前提条件

实例为Redis企业版性能增强型或持久内存型（小版本为1.2.3及以上，升级方法请参见升级小版本）。

注意事项

- 本文的操作对象为企业版实例中的TairString数据。

  > **说明** 企业版实例中可同时设置Redis String（即Redis原生String）和TairString，本文的命令无法对Redis String使用。

- 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新。

  > **说明** 如果您的实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，否则可能出现命令无法识别的情况。

命令列表

TairString命令

命令	语法	简介
EXSET	`EXSET key value [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version]`	若key不存在，则创建新的key，并将value保存到key中；若key已存在，则覆盖原来value的值。
EXGET	`EXGET key`	获取TairString的value和version。
EXSETVER	`EXSETVER key version`	设置目标key的version。
EXINCRBY	`EXINCRBY key num [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version] [MIN minval] [MAX maxval]`	对TairString的value进行自增自减操作，num的范围为long。
EXINCRBYFLOAT	`EXINCRBYFLOAT key num [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version] [MIN minval] [MAX maxval]`	对TairString的value进行自增自减操作，num的范围为double。
EXCAS	`EXCAS key newvalue version`	当目标key的version值与指定的version相等时，则更新key的value值；version不相等，则返回旧的value和version。
EXCAD	`EXCAD key version`	当目标key的version值与指定的version相等时，则删除Key。
DEL	`DEL key [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairString数据。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

EXSET

类别	说明
语法	`EXSET key value [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version]`
时间复杂度	O(1)
命令描述	若key不存在，则创建新的key，并将value保存到key中；若key已存在，则覆盖原来value的值。

类别	说明

选项	• Key：TairString的key，用于指定作为命令调用对象的TairString。
	• value：为key设置的value。
	• EX：指定key的相对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• EXAT：指定key的绝对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• PX：指定key的相对过期时间，单位为毫秒，为0表示马上过期，不传此参数表示不过期。
	• PXAT：指定key的绝对过期时间，单位为毫秒 ，为0表示马上过期，不传此参数表示不过期。
	• NX：只在key不存在时写入。
	• XX：只在key存在时写入。
	• VER：版本号。
	  ◦ 如果key存在，和当前版本号做比较：
	  ▪ 如果相等，写入，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果key不存在或者key当前版本为0，忽略传入的版本号直接设置value，成功后版本号变为1。
	• ABS：绝对版本号。设置后，无论key当前的版本号是多少，完成写入并将key的版本号覆盖为该选项中设置的值。

返回值	• 执行成功：OK。 • 指定了XX且key不存在：nil。 • 指定了NX且key已经存在：nil。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXSET foo bar NX ABS 100`
	返回示例：
	`OK`

EXGET

类别	说明
语法	`EXGET key`
时间复杂度	O(1)
命令描述	获取TairString的value和version。
选项	• Key：TairString的key，用于指定作为命令调用对象的TairString。
返回值	• 执行成功：value与version。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXGET foo`
	返回示例：
	`1) "bar"`
	`2) (integer) 1`

EXSETVER

类别	说明
语法	`EXSETVER key version`
时间复杂度	O(1)
命令描述	设置目标key的version。
选项	• Key：TairString的key，用于指定作为命令调用对象的TairString。 • version：需要设置的版本号。
返回值	• 执行成功：1。 • 若key不存在：0。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXSETVER foo 2`
	返回示例：
	`(integer) 1`

EXINCRBY

类别	说明
语法	`EXINCRBY key num [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version] [MIN minval] [MAX maxval]`
时间复杂度	O(1)
命令描述	对TairString的value进行自增自减操作，num的范围为long。

选项	• Key：TairString的key，用于指定作为命令调用对象的TairString。
	• num：TairString进行自增自减操作的数值，必须为整数。
	• EX：指定key的相对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• EXAT：指定key的绝对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• PX：指定key的相对过期时间，单位为毫秒，为0表示马上过期，不传此参数表示不过期。
	• PXAT：指定key的绝对过期时间，单位为毫秒 ，为0表示马上过期，不传此参数表示不过期。
	• NX：只在key不存在时写入。
	• XX：只在key存在时写入。
	• VER：版本号。
	  ◦ 如果key存在，和当前版本号做比较：
	  ▪ 如果相等，进行自增，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果key不存在或者key当前版本为0，忽略传入的版本号并进行自增操作，成功后版本号变为1。
	• ABS：绝对版本号。设置后，无论key当前的版本号是多少，完成写入并将key的版本号覆盖为该选项中设置的值。
	• MIN：设置TairString value的最小值。
	• MAX：设置TairString value的最大值。

类别	说明
返回值	• 执行成功：操作后value的值。 • 若设置了MAX或MIN，而操作后的valve超过了该范围：(error) ERR increment or decrement would overflow。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXSET foo 1` 命令。
	命令示例：
	`EXINCRBY foo 100 MAX 300`
	返回示例：
	`(integer) 101`

EXINCRBYFLOAT

类别	说明
语法	`EXINCRBYFLOAT` *key num* `[EX` *time*`] [PX` *time*`] [EXAT` *time*`] [PXAT` *time*`] [NX | XX] [VER | ABS` *version*`] [MIN` *minval*`] [MAX` *maxval*`]`
时间复杂度	O(1)
命令描述	对TairString的value进行自增自减操作，num的范围为double。

选项	• Key：TairString的key，用于指定作为命令调用对象的TairString。
	• num：TairString进行自增自减操作的数值，类型为浮点数。
	• EX：指定key的相对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• EXAT：指定key的绝对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• PX：指定key的相对过期时间，单位为毫秒，为0表示马上过期，不传此参数表示不过期。
	• PXAT：指定key的绝对过期时间，单位为毫秒 ，为0表示马上过期，不传此参数表示不过期。
	• NX：只在key不存在时写入。
	• XX：只在key存在时写入。
	• VER：版本号。
	  ◦ 如果key存在，和当前版本号做比较：
	  ▪ 如果相等，进行自增，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果key不存在或者key当前版本为0，忽略传入的版本号并进行自增操作，成功后版本号变为1。
	• ABS：绝对版本号。设置后，无论key当前的版本号是多少，完成写入并将key的版本号覆盖为该选项中设置的值。
	• MIN：设置TairString value的最小值。
	• MAX：设置TairString value的最大值。

返回值	• 执行成功：操作后value的值。 • 若设置了MAX或MIN，而操作后的valve超过了该范围：(error) ERR increment or decrement would overflow。 • 其它情况返回相应的异常信息。

类别	说明

示例	提前执行 `EXSET foo 1` 命令。
	命令示例：
	`EXINCRBYFLOAT foo 10.123`
	返回示例：
	`(integer) 11.123`

EXCAS

类别	说明
语法	`EXCAS key newvalue version`
时间复杂度	O(1)
命令描述	当目标key的version值与指定的version相等时，则更新key的value值；version不相等，则返回旧的value和version。
选项	• Key：TairString的key，用于指定作为命令调用对象的TairString。 • newvalue：若key的version值与指定的version相等，将value修改为newvalue。 • version：用于跟key的现有version值比较的值。
返回值	• 执行成功：["OK", "",最新的version]。中间的""为无意义的空字符串。 • 执行失败：["ERR update version is stale", value, version]。value和version为key当前的value和版本。 • 若key不存在：-1。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXSET foo bar` 命令。
	命令示例：
	`EXCAS foo bzz 1`
	返回示例：
	`1) OK`
	`2)`
	`3) (integer) 2`

EXCAD

类别	说明
语法	`EXCAD key version`
时间复杂度	O(1)
命令描述	当目标key的version值与指定的version相等时，则删除Key。
选项	• Key：TairString的key，用于指定作为命令调用对象的TairString。 • version：用于跟key的现有version值比较的值。

类别	说明
返回值	• 执行成功：1。 • 执行失败：0。 • 若key不存在：-1。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXSET foo bar` 命令。
	命令示例：
	`EXCAD foo 1`
	返回示例：
	`(integer) 1`

3.6.3. CAS和CAD命令

本文介绍云数据库Redis企业版性能增强型实例中新增的String增强类命令，包括CAS和CAD。

使用前提

请注意，本文介绍的命令只有在满足以下条件时才能生效。

- Redis实例为企业版性能增强型。
- 操作对象为性能增强型实例中的Redis String数据。

> **说明** 性能增强型实例中可同时设置Redis String（即Redis原生String）和TairString，CAS和CAD只能对Redis String使用。

命令列表

String增强命令

命令	语法	说明
CAS	CAS <key> <oldvalue> <newvalue>	当oldvalue和key的value相等时，修改value的值为newvalue；不相等则不修改。 **说明** CAS仅适用于操作Redis String类型的数据，如需对TairString做相同的操作，请使用EXCAS。
CAD	CAD <key> <value>	当oldvalue和key的value相等时，删除该key；不相等则不删除。 **说明** CAD仅适用于操作Redis String类型的数据，如需对TairString做相同的操作，请使用EXCAD。

CAS

- 语法
  CAS <key> <oldvalue> <newvalue>
- 时间复杂度
  O(1)
- 命令描述
  CAS（Compare And Set），查看某个key的value是否等于一个指定的值，如果相等，则将value修改为一个新的值；不相等则不修改。
- 参数及选项说明

参数/选项	说明
key	String的key，用于指定作为命令调用对象的String。
oldvalue	用于跟key的现有value比较的值。
newvalue	当oldvalue和key的现有value相等时，将value修改为newvalue。

- 返回值
  - 成功：1。
  - key不存在：-1。
  - 失败：0。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> SET foo bar
OK
127.0.0.1:6379> CAS foo baa bzz
(integer) 0
127.0.0.1:6379> GET foo
"bar"
127.0.0.1:6379> CAS foo bar bzz
(integer) 1
127.0.0.1:6379> GET foo
"bzz"
```

CAD

- 语法
  CAD <key> <value>
- 时间复杂度
  O(1)
- 命令描述
  CAD（Compare And Delete），查看某个key的value是否等于一个指定的值，如果相等，删除该key；不相等则不删除。
- 参数及选项说明

参数/选项	说明
key	String的key，用于指定作为命令调用对象的String。
value	用于跟key的现有value比较的值。

- 返回值
  - 成功：1。
  - key不存在：-1。
  - 失败：0。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> SET foo bar
OK
127.0.0.1:6379> CAD foo bzz
(integer) 0
127.0.0.1:6379> CAD not-exists xxx
(integer) -1
127.0.0.1:6379> CAD foo bar
(integer) 1
127.0.0.1:6379> GET foo
(nil)
```

3.6.4. TairHash

TairHash是一种可为field设置过期时间和版本的Hash类型数据结构，提高了Hash数据结构的灵活性，简化了很多场景下的业务开发工作。

TairHash简介

TairHash不但和Redis Hash一样支持丰富的数据接口和高处理性能，还改变了hash只能为key设置过期时间的限制，可以为field设置过期时间和版本，极大地提高了hash数据结构的灵活性，简化了很多场景下的业务开发工作。TairHash使用高效的Active Expire算法，可以在不对响应时间造成明显影响的前提下，更高效的完成对field的过期判断和删除。

主要特征

- field支持单独设置expire和version。
- field支持高效灵活的主动、被动过期淘汰（expire）策略。
- 语法和原生Redis Hash数据类型类似。

该Module已开源，更多信息请参见TairHash。

前提条件

实例为Tair（Redis企业版）性能增强型。

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

操作对象为性能增强型实例中的TairHash数据。

命令列表

TairHash命令

命令	语法	说明
EXHSET	`EXHSET key field value [EX time] [EXAT time] [PX time] [PXAT time] [NX | XX] [VER | ABS version] [KEEPTTL]`	向key指定的TairHash中插入一个field。如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其值。 **说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。
EXHGET	`EXHGET key field`	获取key指定的TairHash中一个field的值，如果TairHash不存在或者field不存在，则返回nil。
EXHMSET	`EXHMSET key field value [field value ...]`	同时向key指定的TairHash中插入多个field，如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其值。
EXHPEXPIREAT	`EXHPEXPIREAT key field milliseconds-timestamp [VER | ABS version]`	在key指定的TairHash中为一个field设置绝对过期时间，精确到毫秒。

命令	语法	说明
EXHPEXPIRE	`EXHPEXPIRE key field milliseconds [VER | ABS version]`	在key指定的TairHash中为一个field设置相对过期时间，单位为毫秒。
EXHEXPIREAT	`EXHEXPIREAT key field timestamp [VER | ABS version]`	在key指定的TairHash中为一个field设置绝对过期时间，精确到秒。
EXHEXPIRE	`EXHEXPIRE key field seconds [VER | ABS version]`	在key指定的TairHash中为一个field设置相对过期时间，单位为秒。
EXHPTTL	`EXHPTTL key field`	查看key指定的TairHash中一个field的剩余过期时间，结果精确到毫秒。
EXHTTL	`EXHTTL key field`	查看key指定的TairHash中一个field的过期时间，结果精确到秒。
EXHVER	`EXHVER key field`	查看key指定的TairHash中一个field的当前版本号。
EXHSETVER	`EXHSETVER key field version`	设置key指定的TairHash中一个field的版本号。
EXHINCRBY	`EXHINCRBY key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`	将key指定的TairHash中一个field的value增加num，num为一个整数。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前插入该field并将其值设置为0。 **说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。
EXHINCRBYFLOAT	`EXHINCRBYFLOAT key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`	将key指定的TairHash中一个field的value增加num，num为一个浮点数。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前插入该field并将其值设置为0。 **说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。
EXHGETWITHVER	`EXHGETWITHVER key field`	同时获取key指定的TairHash一个field的值和版本，如果TairHash不存在或者field不存在，则返回nil。
EXHMGET	`EXHMGET key field [field ...]`	同时获取key指定的TairHash多个field的值，如果TairHash不存在或者field不存在，则返回nil。
EXHMGETWITHVER	`EXHMGETWITHVER key field [field ...]`	同时获取key指定的TairHash多个field的值和版本。
EXHLEN	`EXHLEN key [NOEXP]`	获取key指定的TairHash中field个数，该命令不会触发对过期field的被动淘汰，也不会将其过滤掉，所以结果中可能包含已经过期但还未被删除的field。如果只想返回当前没有过期的field个数，可以在命令中设置NOEXP选项。
EXHEXISTS	`EXHEXISTS key field`	查询key指定的TairHash中是否存在对应的field。
EXHSTRLEN	`EXHSTRLEN key field`	获取key指定的TairHash中一个field对应的value的长度。
EXHKEYS	`EXHKEYS key`	获取key指定的TairHash中所有的field，该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。

命令	语法	说明
EXHVALS	`EXHVALS key`	获取key指定的TairHash中所有field的值，该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。
EXHGETALL	`EXHGETALL key`	获取key指定的TairHash中所有field及其value，该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。
EXHSCAN	`EXHSCAN key op subkey [MATCH pattern] [COUNT count]`	扫描key指定的TairHash。扫描的方式（使用op选项设置）可以为>、>=、<、<=、==、^、$。扫描的同时可以根据MATCH指定的pattern对subkey进行正则过滤，还可以使用COUNT对结果数目进行限制，如果不指定则默认值为10。该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。
EXHDEL	`EXHDEL key field [field ...]`	删除key指定的TairHash中的一个field，如果TairHash不存在或者field不存在则返回0，成功删除返回1。
DEL	`DEL <key> [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairHash数据。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

EXHSET

类别	说明
语法	`EXHSET key field value [EX time] [EXAT time] [PX time] [PXAT time] [NX | XX] [VER | ABS version] [KEEPTTL]`
时间复杂度	O(1)
命令描述	向key指定的TairHash中插入一个field。如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其值。 **说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。

类别	说明

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• field：TairHash中的一个元素，一个TairHash key可以有多个field。
	• value：field对应的值，一个field只能有一个value。
	• EX：指定field的相对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• EXAT：指定field的绝对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• PX：指定field的相对过期时间，单位为毫秒，为0表示马上过期，不传此参数表示不过期。
	• PXAT：指定field的绝对过期时间，单位为毫秒 ，为0表示马上过期，不传此参数表示不过期。
	• NX：只在field不存在时插入。
	• XX：只在field存在时插入。
	• VER：版本号。
	  ◦ 如果field存在，和当前版本号做比较：
	  ▪ 如果相等，继续操作，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。
	• ABS：绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。
	• KEEPTTL：在不指定*EX*、*EXAT*、*PX*或*PXAT*选项时，使用KEEPTTL选项会保留field当前的过期设置。
	说明 若不使用KEEPTTL选项，EXHSET命令会默认删除field上原先设置的过期时间。

返回值	• 新建field并成功为它设置值：1。 • field已经存在，成功覆盖旧值：0。 • 指定了XX且field不存在：-1。 • 指定了NX且field已经存在：-1。 • 指定了VER且版本和当前版本不匹配："ERR update version is stale"。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHSET myhash field1 val EX 10`
	返回示例：
	`(integer) 1`

EXHGET

类别	说明
语法	`EXHGET key field`
时间复杂度	O(1)
命令描述	获取key指定的TairHash中一个field的值，如果TairHash不存在或者field不存在，则返回nil。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。

类别	说明
返回值	• field存在且操作成功：field对应的值。 • key不存在或者field不存在：nil。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXHSET myhash field1 val` 命令。
	命令示例：
	`EXHGET myhash field1`
	返回示例：
	`"val"`

EXHMSET

类别	说明
语法	`EXHMSET key field value [field value ...]`
时间复杂度	O(N)
命令描述	同时向key指定的TairHash中插入多个field，如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其值。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。 • value：field对应的值，一个field只能有一个value。
返回值	• 成功：OK。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHMSET myhash field1 val1 field2 val2`
	返回示例：
	`OK`

EXHPEXPIREAT

类别	说明
语法	`EXHPEXPIREAT key field milliseconds-timestamp [VER | ABS version]`
时间复杂度	O(1)
命令描述	在key指定的TairHash中为一个field设置绝对过期时间，精确到毫秒。

类别	说明

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• field：TairHash中的一个元素，一个TairHash key可以有多个field。
	• milliseconds-timestamp：精确到毫秒的UNIX 时间戳 （Unix timestamp）。
	• VER：版本号。
	  ◦ 如果field存在，和当前版本号做比较：
	  ▪ 如果相等，继续操作，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。
	• ABS：绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。

返回值	• field存在且设置成功：1。 • field不存在：0。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHPEXPIREAT myhash field1 1293840000`
	返回示例：
	`(integer) 1`

EXHPEXPIRE

类别	说明
语法	`EXHPEXPIRE key field milliseconds [VER | ABS version]`
时间复杂度	O(1)
命令描述	在key指定的TairHash中为一个field设置相对过期时间，单位为毫秒。

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• field：TairHash中的一个元素，一个TairHash key可以有多个field。
	• milliseconds：相对过期时间，单位为毫秒。
	• VER：版本号。
	  ◦ 如果field存在，和当前版本号做比较：
	  ▪ 如果相等，继续操作，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。
	• ABS：绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。

返回值	• field存在且设置成功：1。 • field不存在：0。 • 其它情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`EXHPEXPIRE myhash field1 1000`
	返回示例：
	`(integer) 1`

EXHEXPIREAT

类别	说明
语法	`EXHEXPIREAT key field timestamp [VER | ABS version]`
时间复杂度	O(1)
命令描述	在key指定的TairHash中为一个field设置绝对过期时间，精确到秒。

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• field：TairHash中的一个元素，一个TairHash key可以有多个field。
	• timestamp：精确到秒的UNIX 时间戳 （Unix timestamp）。
	• VER：版本号。
	  ◦ 如果field存在，和当前版本号做比较：
	  ▪ 如果相等，继续操作，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。
	• ABS：绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。

返回值	• field存在且设置成功：1。 • field不存在：0。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHEXPIREAT myhash field1 1293840000`
	返回示例：
	`(integer) 1`

EXHEXPIRE

类别	说明
语法	`EXHEXPIRE key field seconds [VER | ABS version]`
时间复杂度	O(1)
命令描述	在key指定的TairHash中为一个field设置相对过期时间，单位为秒。

类别	说明

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• field：TairHash中的一个元素，一个TairHash key可以有多个field。
	• seconds：相对过期时间，单位为秒。
	• VER：版本号。
	  ◦ 如果field存在，和当前版本号做比较：
	  ▪ 如果相等，继续操作，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。
	• ABS：绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。

返回值	• field存在且设置成功：1。 • field不存在：0。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHEXPIRE myhash field1 100`
	返回示例：
	`(integer) 1`

EXHPTTL

类别	说明
语法	`EXHPTTL key field`
时间复杂度	O(1)
命令描述	查看key指定的TairHash中一个field的剩余过期时间，结果精确到毫秒。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• key或者field不存在：-2。 • field存在但是没有设置过期时间：-1。 • field存在且设置了过期时间：过期时间，单位为毫秒。 • 其它情况返回相应的异常信息。

示例	提前执行`EXHSET myhash field1 val1 EX 100`命令。
	命令示例：
	`EXHPTTL myhash field1`
	返回示例：
	`(integer) 97213`

EXHTTL

类别	说明
语法	`EXHTTL key field`
时间复杂度	O(1)
命令描述	查看key指定的TairHash中一个field的过期时间，结果精确到秒。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• key或者field不存在：-2。 • field存在但是没有设置过期时间：-1。 • field存在且设置了过期时间：过期时间，单位为秒。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXHSET myhash field1 val1 EX 100` 命令。
	命令示例：
	`EXHTTL myhash field1`
	返回示例：
	`(integer) 85`

EXHVER

类别	说明
语法	`EXHVER key field`
时间复杂度	O(1)
命令描述	查看key指定的TairHash中一个field的当前版本号。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• key不存在：-1。 • field不存：-2。 • 查询成功：field的版本号。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHVER myhash field1`
	返回示例：
	`(integer) 1`

EXHSETVER

类别	说明
语法	`EXHSETVER key field version`

类别	说明
时间复杂度	O(1)
命令描述	设置key指定的TairHash中一个field的版本号。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• TairHash或者field不存在：0。 • 设置成功：1。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHSETVER myhash field1 3`
	返回示例：
	`(integer) 1`

EXHINCRBY

类别	说明
语法	`EXHINCRBY key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`
时间复杂度	O(1)
命令描述	将key指定的TairHash中一个field的value增加num，num为一个整数。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前插入该field并将其值设置为0。 **说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• field：TairHash中的一个元素，一个TairHash key可以有多个field。
	• num：需要为field的value增加的整数值。
	• EX：指定field的相对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• EXAT：指定field的绝对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• PX：指定field的相对过期时间，单位为毫秒，为0表示马上过期，不传此参数表示不过期。
	• PXAT：指定field的绝对过期时间，单位为毫秒 ，为0表示马上过期，不传此参数表示不过期。
	• VER：版本号。
	  ◦ 如果field存在，和当前版本号做比较：
	  ▪ 如果相等，继续操作，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。
	• ABS：绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。
	• MIN：value的最小值，小于该值则提示异常。
	• MAX：value的最大值，大于该值则提示异常。
	• KEEPTTL：在不指定*EX*、*EXAT*、*PX*或*PXAT*选项时，使用KEEPTTL选项会保留field当前的过期设置。

类别	说明
返回值	• 成功：与num相加后value的值。 • 其它情况返回异常。

示例	提前执行 `EXHMSET myhash field1 10` 命令。
	命令示例：
	`EXHINCRBY myhash field1 100`
	返回示例：
	`(integer) 110`

EXHINCRBYFLOAT

类别	说明
语法	`EXHINCRBYFLOAT key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`
时间复杂度	O(1)
命令描述	将key指定的TairHash中一个field的value增加num，num为一个浮点数。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前插入该field并将其值设置为0。 **说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• field：TairHash中的一个元素，一个TairHash key可以有多个field。
	• num：需要为field的value增加的值，类型为浮点。
	• EX：指定field的相对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• EXAT：指定field的绝对过期时间，单位为秒，为0表示马上过期，不传此参数表示不过期。
	• PX：指定field的相对过期时间，单位为毫秒，为0表示马上过期，不传此参数表示不过期。
	• PXAT：指定field的绝对过期时间，单位为毫秒 ，为0表示马上过期，不传此参数表示不过期。
	• VER：版本号。
	  ◦ 如果field存在，和当前版本号做比较：
	  ▪ 如果相等，继续操作，且版本号加1。
	  ▪ 如果不相等，返回异常。
	  ◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。
	• ABS：绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。
	• MIN：value的最小值，小于该值则提示异常。
	• MAX：value的最大值，大于该值则提示异常。
	• KEEPTTL：在不指定*EX*、*EXAT*、*PX*或*PXAT*选项时，使用KEEPTTL选项会保留field当前的过期设置。

返回值	• 成功：与num相加后value的值。 • 其它情况返回异常。

类别	说明

示例	提前执行 `EXHMSET myhash field1 10` 命令。
	命令示例：
	`EXHINCRBYFLOAT myhash field1 9.235`
	返回示例：
	`"19.235"`

EXHGETWITHVER

类别	说明
语法	`EXHGETWITHVER key field`
时间复杂度	O(1)
命令描述	同时获取key指定的TairHash一个field的值和版本，如果TairHash不存在或者field不存在，则返回nil。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• field存在且操作成功：field对应的值和版本。 • key不存在或者field不存在：nil。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHGETWITHVER myhash field1`
	返回示例：
	`1) "19.235"`
	`2) (integer) 5`

EXHMGET

类别	说明
语法	`EXHMGET key field [field ...]`
时间复杂度	O(1)
命令描述	同时获取key指定的TairHash多个field的值，如果TairHash不存在或者field不存在，则返回nil。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• key不存在：nil。 • key存在且查询的所有field都存在：返回一个数组，数组的每一个元素对应一个field的value。 • key存在且查询的field中有不存在的：返回一个数组，数组的每一个元素对应一个field的value，不存在的field对应的元素显示为nil。 • 其它情况返回相应的异常信息。

类别	说明

示例	提前执行 `EXHMSET myhash field1 10 field2 var1` 命令。
	命令示例：
	`EXHMGET myhash field1 field2`
	返回示例：
	`1) "10"`
	`2) "var1"`

EXHMGETWITHVER

类别	说明
语法	`EXHMGETWITHVER key field [field ...]`
时间复杂度	O(1)
命令描述	同时获取key指定的TairHash多个field的值和版本。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• key不存在：nil。 • key存在且查询的所有field都存在：返回一个数组，数组的每一个元素对应一个field的value和version。 • key存在且查询的field中有不存在的：返回一个数组，数组的每一个元素对应一个field的value和version，不存在的field对应的元素显示为nil。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXHMSET myhash field1 10 field2 var1` 命令。
	命令示例：
	`EXHMGETWITHVER myhash field1 field2`
	返回示例：
	`1) 1) "10"`
	`2) (integer) 1`
	`2) 1) "var1"`
	`2) (integer) 1`

EXHLEN

类别	说明
语法	`EXHLEN key [NOEXP]`
时间复杂度	未设置*NOEXP*选项时是O(1)，设置*NOEXP*选项时是O(N)。
命令描述	获取key指定的TairHash中field个数，该命令不会触发对过期field的被动淘汰，也不会将其过滤掉，所以结果中可能包含已经过期但还未被删除的field。如果只想返回当前没有过期的field个数，可以在命令中设置NOEXP选项。

类别	说明

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• NOEXP：该命令默认不会触发对过期field的被动淘汰，也不会将其过滤掉，所以结果中可能包含已经过期但还未被删除的field。如果只想返回当前没有过期的field个数，可以在命令中设置*NOEXP*选项。在设置*NOEXP*时：
	  ○ 因为要遍历整条TairHash数据，**EXHLEN**命令的响应时间将受到Tairhash大小的影响。
	  ○ **EXHLEN**命令的返回结果中会过滤掉过期的field，但过期field不会被淘汰。

返回值	• key不存在或者field不存在：0。 • 成功：field个数。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHLEN myhash`
	返回示例：
	`(integer) 2`

EXHEXISTS

类别	说明
语法	`EXHEXISTS key field`
时间复杂度	O(1)
命令描述	查询key指定的TairHash中是否存在对应的field。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• key不存在或者field不存在：0。 • field存在：1。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHEXISTS myhash field1`
	返回示例：
	`(integer) 1`

EXHSTRLEN

类别	说明
语法	`EXHSTRLEN key field`
时间复杂度	O(1)
命令描述	获取key指定的TairHash中一个field对应的value的长度。

类别	说明
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。
返回值	• key不存在或者field不存在：0。 • 查询成功：value的长度。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHSTRLEN myhash field1`
	返回示例：
	`(integer) 2`

EXHKEYS

类别	说明
语法	`EXHKEYS key`
时间复杂度	O(N)
命令描述	获取key指定的TairHash中所有的field，该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
返回值	• key不存在：返回一个空数组。 • key存在：返回一个数组，数组的每一位对应TairHash中的每一个field。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXHMSET myhash field1 10 field2 var1` 命令。
	命令示例：
	`EXHKEYS myhash`
	返回示例：
	`1) "field1"`
	`2) "field2"`

EXHVALS

类别	说明
语法	`EXHVALS key`
时间复杂度	O(N)
命令描述	获取key指定的TairHash中所有field的值，该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。

类别	说明
返回值	• key不存在：返回一个空数组。 • key存在：返回一个数组，数组的每个元素对应TairHash中的一个field的value。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXHMSET myhash field1 10 field2 var1` 命令。
	命令示例：
	`EXHVALS myhash`
	返回示例：
	`1) "10"`
	`2) "var1"`

EXHGETALL

类别	说明
语法	`EXHGETALL key`
时间复杂度	O(N)
命令描述	获取key指定的TairHash中所有field及其value，该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
返回值	• key不存在：返回一个空数组。 • key存在：返回一个数组，数组的每个元素对应TairHash中的一对field和value。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXHMSET myhash field1 10 field2 var1` 命令。
	命令示例：
	`EXHGETALL myhash`
	返回示例：
	`1) "field1"`
	`2) "10"`
	`3) "field2"`
	`4) "var1"`

EXHSCAN

类别	说明
语法	`EXHSCAN key op subkey [MATCH pattern] [COUNT count]`
时间复杂度	每次调用时是O(1)，遍历整个TairHash时是O(N)。
命令描述	扫描key指定的TairHash。扫描的方式（使用op选项设置）可以为>、>=、<、<=、==、^、$。扫描的同时可以根据MATCH指定的pattern对subkey进行正则过滤，还可以使用COUNT对结果数目进行限制，如果不指定则默认值为10。该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。

类别	说明

选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。
	• op：用于定位扫描的起点，可选值如下。
	  ◦ >：表示从第一个大于subkey的field开始。
	  ◦ >=：表示从第一个大于等于subkey的field开始。
	  ◦ <：表示从第一个小于subkey的field开始。
	  ◦ <=：表示从第一个小于等于subkey的field开始。
	  ◦ ==：表示从第一个等于subkey的field开始。
	  ◦ ^：表示从第一个field开始。
	  ◦ $：表示从最后一个field开始。
	• subkey：用于与op选项搭配，设置扫描起始位置，当op为^或$时该值将被忽略。
	• MATCH：用于过滤扫描结果。
	• COUNT：用于规定单次扫描field的个数（默认为10）。
	说明 COUNT仅表示每次扫描TairHash的field的个数，不代表最终一定会返回COUNT个field结果集，结果集的大小还要根据TairHash中当前field个数和是否指定MATCH进行过滤而定。

返回值	• key不存在：返回一个空数组。
	• key存在：返回一个具有两个元素的数组：
	  ◦ 第一个元素：下一次扫描的起始field，如果该key已扫描完成，则该元素为空。
	  ◦ 第二个元素：本次扫描结果，包含field和value。
	• 其它情况返回相应的异常信息。

示例	提前执行 `EXHMSET myhashkey field1 val1 field2 val2 field3 val3 field4 val4 field5 val5` 命令。
	命令示例：
	`EXHSCAN myhashkey ^ xx COUNT 3`
	返回示例：
	`1) "field4"`
	`2) 1) "field1"`
	`2) "val1"`
	`3) "field2"`
	`4) "val2"`
	`5) "field3"`
	`6) "val3"`

EXHDEL

类别	说明
语法	`EXHDEL key field [field ...]`
时间复杂度	O(1)
命令描述	删除key指定的TairHash中的一个field，如果TairHash不存在或者field不存在则返回0 ，成功删除返回1。
选项	• Key：TairHash的key，用于指定作为命令调用对象的TairHash。 • field：TairHash中的一个元素，一个TairHash key可以有多个field。

类别	说明
返回值	• key不存在或者field不存在：0。 • 删除成功：1。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXHDEL myhash field1`
	返回示例：
	`(integer) 1`

3.6.5. TairGIS

TairGis是一种使用R-Tree做索引，支持地理信息系统GIS（Geographic Information System）相关接口的数据结构。Redis的原生GEO命令是使用GeoHash和Redis Sorted Set结构完成的，主要用于点的查询，TairGIS在此基础上还支持线、面的查询，功能更加强大。

主要特性

- 使用R-Tree作为索引存储。
- 支持线、面的相关查询（含相交查询）。
- 通过GIS.SEARCH可实现原生Redis `GEORADIUS` 命令的功能。

最佳实践

基于TairGIS轻松实现用户轨迹监测和电子围栏

前提条件

实例为Tair（Redis企业版）性能增强型。

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

操作对象为性能增强型实例中的TairGIS数据。

命令列表

TairGIS命令

命令	语法	说明
GIS.ADD	`GIS.ADD area polygonName polygonWkt [polygonName polygonWkt ...]`	在area中添加指定名称的多边形（可添加多个），使用WKT（Well-known text）描述。 **说明** WKT是一种文本标记语言，用于描述矢量几何对象、空间参照系统及空间参照系统之间的转换。
GIS.GET	`GIS.GET area polygonName`	获取目标area中指定多边形的WKT信息。
GIS.GETALL	`GIS.GETALL area [WITHOUTWKT]`	获取目标area中所有多边形的名称和WKT信息。如果设置了WITHOUTWKT选项，仅返回多边形的名称。
GIS.CONTAINS	`GIS.CONTAINS area polygonWkt [WITHOUTWKT]`	判断指定的点、线或面是否包含在目标area的多边形中，若包含，则返回目标area中命中的多边形数量与多边形信息。

命令	语法	说明
GIS.WITHIN	`GIS.WITHIN area polygonWkt [WITHOUTWKT]`	判断目标area是否包含在指定的点、线或面中，若包含，则返回目标area中命中的多边形数量与多边形信息。
GIS.INTERSECTS	`GIS.INTERSECTS area polygonWkt`	判断指定的点、线或面与目标area的多边形是否相交，若相交，则返回目标area中与其相交的多边形数量与多边形信息。
GIS.SEARCH	`GIS.SEARCH area [RADIUS longitude latitude distance M|KM|FT|MI] [MEMBER field distance M|KM|FT|MI] [GEOM geom] [COUNT count] [ASC|DESC] [WITHDIST] [WITHOUTWKT]`	在指定经、纬度及半径距离范围内，查找目标area中的点。
GIS.DEL	`GIS.DEL area polygonName`	删除目标area中指定的多边形。
DEL	`DEL key [key ...]`	原生Redis命令，可以删除一条或多条TairGIS数据。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

GIS.ADD

类别	说明
语法	`GIS.ADD area polygonName polygonWkt [polygonName polygonWkt ...]`
时间复杂度	O(log n)
命令描述	在area中添加指定名称的多边形（可添加多个），使用WKT（Well-known text）描述。 **说明** WKT是一种文本标记语言，用于描述矢量几何对象、空间参照系统及空间参照系统之间的转换。

选项	• area：一个几何概念。
	• PolygonName：多边形的名称。
	• polygonWkt：多边形的描述信息，表示现实世界的经、纬度，使用WKT（Well-known text）描述，支持如下类型。
	  ◦ POINT：描述一个点的WKT信息，例如 `'POINT (120.086631 30.138141)'`，表示该POINT位于经度120.086631，纬度30.138141。
	  ◦ LINESTRING：描述一条线的WKT信息，由两个POINT组成，例如 `'LINESTRING (30 10, 40 40)'`。
	  ◦ POLYGON：描述一个多边形的WKT信息，由多个POINT组成，例如 `'POLYGON ((31 20, 29 20, 29 21, 31 31))'`。
	说明
	◦ 经度的取值范围为(-180,180)， 纬度的取值范围为(-90,90)。
	◦ 不支持如下集合类型：MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRY和COLLECTION。

类别	说明
返回值	• 执行成功：返回插入和更新成功的多边形数量。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'`
	返回示例：
	`(integer) 1`

GIS.GET

类别	说明
语法	`GIS.GET area polygonName`
时间复杂度	O(1)
命令描述	获取目标area中指定多边形的WKT信息。
选项	• area：一个几何概念。 • PolygonName：多边形的名称。
返回值	• 执行成功：WKT信息。 • area或polygonName不存在：nil。 • 其它情况返回相应的异常信息。

示例	提前执行`GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'`命令。
	命令示例：
	`GIS.GET hangzhou campus`
	返回示例：
	`'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'`

GIS.GETALL

类别	说明
语法	`GIS.GETALL area [WITHOUTWKT]`
时间复杂度	O(n)
命令描述	获取目标area中所有多边形的名称和WKT信息。如果设置了WITHOUTWKT选项，仅返回多边形的名称。
选项	• area：一个几何概念。 • WITHOUTWKT：用于控制是否返回多边形的WKT信息，如果加上该参数，则不返回多边形的WKT信息。
返回值	• 执行成功：返回多边形名称和WKT信息，如果设置了WITHOUTWKT选项，仅返回多边形的名称。 • area不存在：nil。 • 其它情况返回相应的异常信息。

类别	说明

示例	提前执行 `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` 命令。
	命令示例：
	`GIS.GETALL hangzhou`
	
	返回示例：
	`1) "campus"`
	`2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.CONTAINS

类别	说明
语法	`GIS.CONTAINS area polygonWkt [WITHOUTWKT]`
时间复杂度	• 最理想情况：$O(\log_M n)$ • 最差情况：log(n)
命令描述	判断指定的点、线或面是否包含在目标area的多边形中，若包含，则返回目标area中命中的多边形数量与多边形信息。

选项	• area：一个几何概念。
	• polygonWkt：指定与目标area进行比较的多边形描述信息，使用WKT（Well-known text）描述，支持如下类型。
	  ◦ POINT：描述一个点的WKT信息。
	  ◦ LINESTRING：描述一条线的WKT信息。
	  ◦ POLYGON：描述一个多边形的WKT信息。
	• WITHOUTWKT：用于控制是否返回多边形的WKT信息，如果加上该参数，则不返回多边形的WKT信息。

返回值	• 执行成功：命中的多边形数量与多边形信息。 • area不存在：empty list or set。 • 其它情况返回相应的异常信息。

示例	提前执行 `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` 命令。
	命令示例：
	`GIS.CONTAINS hangzhou 'POINT (30 11)'`
	
	返回示例：
	`1) "1"`
	`2) 1) "campus"`
	`   2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.WITHIN

类别	说明
语法	`GIS.WITHIN area polygonWkt [WITHOUTWKT]`

类别	说明
时间复杂度	• 最理想情况：$O(\log_M n)$ • 最差情况：log(n)
命令描述	判断目标area是否包含在指定的点、线或面中，若包含，则返回目标area中命中的多边形数量与多边形信息。

选项	• area：一个几何概念。
	• polygonWkt：指定与目标area进行比较的多边形描述信息，使用WKT（Well-known text）描述，支持如下类型。
	  ◦ POINT：描述一个点的WKT信息。
	  ◦ LINESTRING：描述一条线的WKT信息。
	  ◦ POLYGON：描述一个多边形的WKT信息。
	  说明 不支持MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRY和COLLECTION。
	• WITHOUTWKT：用于控制是否返回多边形的WKT信息，如果加上该参数，则不返回多边形的WKT信息。

返回值	• 执行成功：命中的多边形数量与多边形信息。 • area不存在：empty list or set。 • 其它情况返回相应的异常信息。

示例	提前执行 `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` 命令。
	命令示例：
	`GIS.WITHIN hangzhou 'POLYGON ((30 5, 50 50, 20 50, 5 20, 30 5))'`
	返回示例：
	`1) "1"`
	`2) 1) "campus"`
	`2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.INTERSECTS

类别	说明
语法	`GIS.INTERSECTS area polygonWkt`
时间复杂度	• 最理想情况：$O(\log_M n)$ • 最差情况：log(n)
命令描述	判断指定的点、线或面与目标area的多边形是否相交，若相交，则返回目标area中与其相交的多边形数量与多边形信息。

选项	• area：一个几何概念。
	• polygonWkt：指定与目标area进行比较的多边形描述信息，使用WKT（Well-known text）描述，支持如下类型。
	  ◦ POINT：描述一个点的WKT信息。
	  ◦ LINESTRING：描述一条线的WKT信息。
	  ◦ POLYGON：描述一个多边形的WKT信息。
	• WITHOUTWKT：用于控制是否返回多边形的WKT信息，如果加上该参数，则不返回多边形的WKT信息。

类别	说明
返回值	• 执行成功：命中的多边形数量与多边形信息。 • area不存在：empty list or set。 • 其它情况返回相应的异常信息。

示例	提前执行 `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` 命令。
	命令示例：
	`GIS.INTERSECTS hangzhou 'LINESTRING (30 10, 40 40)'`
	返回示例：
	`1) "1"`
	`2) 1) "campus"`
	`2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.SEARCH

类别	说明

语法	`GIS.SEARCH area [RADIUS longitude latitude distance M|KM|FT|MI]`
	`[MEMBER field distance M|KM|FT|MI]`
	`[GEOM geom]`
	`[COUNT count]`
	`[ASC|DESC]`
	`[WITHDIST]`
	`[WITHOUTWKT]`

时间复杂度	• 最理想情况：$O(\log_M n)$ • 最差情况：log(n)
命令描述	在指定经、纬度及半径距离范围内，查找目标area中的点。
选项	• area：一个几何概念。 • RADIUS：传入经度（longitude）、纬度（latitude）、半径距离（distance）和半径单位（M表示米、KM表示千米、FT表示英尺、MI表示英里）进行搜索，例如 `RADIUS 15 37 200 KM` 。 • MEMBER：选择当前area中已存在的POINT作为搜索原点，并指定半径进行搜索，取值顺序为多边形名称（field）、半径（distance）、半径单位（M表示米、KM表示千米、FT表示英尺、MI表示英里），例如 `MEMBER Agrigento 100 KM` 。 • GEOM：按照WKT的格式设置搜索范围，可以是任意多边形，例如 `GEOM 'POLYGON((10 30,20 30,20 40,10 40))'` 。 • COUNT：用于限定返回的个数，例如 `COUNT 3` 。 • ASC|DESC：用于控制返回信息按照距离排序，ASC表示根据中心位置，由近到远排序；DESC表示由远到近排序。 • WITHDIST：用于控制是否返回目标点与搜索原点的距离。 • WITHOUTWKT：用于控制是否返回目标点的WKT信息，如果加上该参数，则不返回WKT信息。 **说明** 只能同时使用RADIUS、MEMBER和GEOM中的一种方式。
返回值	• 执行成功：命中的目标点数量与WKT信息。 • area不存在：empty list or set。 • 其它情况返回相应的异常信息。

类别	说明

示例	提前执行 `GIS.ADD Sicily "Palermo" "POINT (13.361389 38.115556)" "Catania" "POINT(15.087269 37.502669)"` 命令。
	命令示例：
	`GIS.SEARCH Sicily RADIUS 15 37 200 km WITHDIST ASC`
	返回示例：
	`1) (integer) 2`
	`2) 1) "Catania"`
	`2) "POINT(15.087269 37.502669)"`
	`3) "56.4413"`
	`4) "Palermo"`
	`5) "POINT(13.361389 38.115556)"`
	`6) "190.4424"`

GIS.DEL

类别	说明
语法	`GIS.DEL area polygonName`
时间复杂度	O(log n)
命令描述	删除目标area中指定的多边形。
选项	• area：一个几何概念。 • PolygonName：多边形的名称。
返回值	• 执行成功：OK。 • area或polygonName不存在：nil。 • 其它情况返回相应的异常信息。

示例	提前执行 `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` 命令。
	命令示例：
	`GIS.DEL hangzhou campus`
	返回示例：
	`OK`

3.6.6. TairBloom

TairBloom首先是一种概率性数据结构（space-efficient probabilistic data structure），主要使用较低的内存消耗来判断一个元素是否存在；其次，TairBloom具有动态扩容的能力，可在扩容的同时维持误判率的稳定。

TairBloom简介

TairBloom首先是一种概率性数据结构（space-efficient probabilistic data structure），主要使用较低的内存消耗来判断一个元素是否存在；其次，TairBloom具有动态扩容的能力，可在扩容的同时维持误判率的稳定。

在传统的Redis数据结构中，可以使用Hash、Set、String的Bitset等实现类似功能，但这些实现方式不是内存占用量非常大，就是无法动态伸缩和保持误判率不变。因此，TairBloom非常适合需要高效判断大量数据是否存在且允许一定误判率的业务场景。业务可以直接使用TairBloom提供的Bloom Filter（布隆过滤器）接口，无需二次封装，更无需在本地实现布隆过滤器的功能。

主要特性

- 内存占用低。

- 可动态扩容。
- 可自定义的误判率（False Positive Rate）且在扩容时保持不变。

典型场景

适用于直播、音乐、电商等行业的推荐系统或爬虫系统等，例如：

- 推荐系统：将用户读过的文章通过TairBloom记录，并在给用户推荐新文章前进行查询，实现给用户推荐感兴趣，且没读过的文章。
- 爬虫系统：在面对海量的URL时，将已经爬取过的URL进行过滤、去重操作，减少重复爬取的无效工作量。

最佳实践

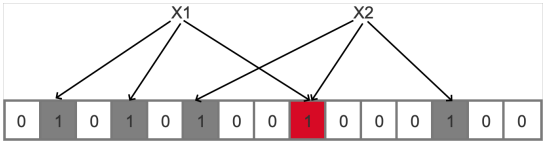
基于TairBloom打造推荐系统 》

基于TairBloom优化爬虫系统 》

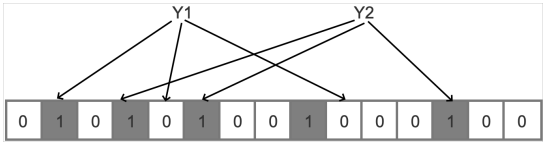
原理介绍

TairBloom作为一种Scalable Bloom Filter的实现，具有动态扩容的能力，同时误判率（False Positive Rate）维持不变。而Scalable Bloom Filter是在Bloom Filter的基础上进行优化的，下文将简单介绍Bloom Filter与Scalable Bloom Filter的基本原理。

- Bloom Filter
  布隆过滤器是一个高空间利用率的概率性数据结构，由Burton Bloom于1970年提出，用于测试一个元素是否在集合中。
  新创建的布隆过滤器是一串被置为0的Bit数组（假设有m位），同时声明k个不同的Hash函数生成统一的随机分布（k是一个小于m的常数）。向布隆过滤器中添加元素时，通过k个Hash函数将元素映射到Bit中的k个点，并将这些位置的值设置为1，一个Bit位可能被不同数据共享。下图展示了假设布隆过滤器的k为3，向其插入X1、X2的过程。

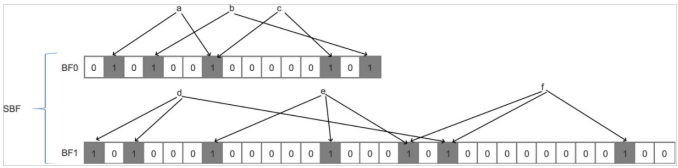


  查询元素时，仍通过k个Hash函数得到对应的k个位，判断目标位置是否为1，若目标位置全为1则认为该元素在布隆过滤器内，否则认为该元素不存在，下图展示了在布隆过滤器中查询Y1和Y2是否存在的过程。



  由上图可以发现，虽然从未向布隆过滤器中插入过Y2这个元素，但是布隆过滤器却判断Y2存在，因此，布隆过滤器是可能存在误判的，即存在假阳性（false positive）。至此，可以得出关于布隆过滤器的几个特性：
  - Bit位可能被不同数据共享。
  - 存在假阳性（false positive），且布隆过滤器中的元素越多，假阳性的可能性越大，但不存在假阴性（false negative），即不会将存在的元素误判为不存在。
  - 元素可以被加入布隆过滤器，但无法被删除，因为Bit位是可以共享的，删除时有可能会影响到其他元素。

- Scalable Bloom Filter
  随着布隆过滤器中添加的元素越来越多，误判率也越来越高，若希望误判率稳定不变，需同步增加布隆过滤器的大小，但是布隆过滤器由于结构限制无法进行扩容。因此，Scalable Bloom Filter提出创建新的布隆过滤器，将多个布隆过滤器组装成一个布隆过滤器使用。
  下图展示了一个Scalable Bloom Filter的基本模型（下文简称SBF）。该SBF一共包含BF0和BF1两层。在一开始，SBF只包含BF0层，假设在插入a、b、c三个元素后，BF0层已经无法保证用户设定的误判率，此时会创建新的一层（BF1层）进行扩容。因此，后面的d、e、f元素会插入到BF1层中。同理，当BF1层也无法满足误判率时，会创建新的一层（BF2层），如此进行下去。



  Scalable Bloom Filter只会向最后一层插入数据，同时也从最后一层开始查询，直到查询至BF0层。更多信息，请参见Scalable Bloom Filter。

前提条件

实例为Tair（Redis企业版）性能增强型。

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

- 操作对象为性能增强型实例中的TairBloom数据。
- 提前规划好初始容量与错误率，若目标key的预计容量远大于100，请通过 `BF.RESERVE` 创建TairBloom，不建议直接执行 `BF.ADD` 命令。
  直接执行 `BF.ADD` 与执行 `BF.RESERVE` 的区别如下。
  - `BF.ADD`（或 `BF.MADD`）：执行时若目标key不存在，Tair会自动创建TairBloom，默认容量（capacity）为100，错误率（error_rate）为0.01。若您的容量远远大于100，后续仅能通过扩容增加元素。
    而TairBloom的扩容是通过增加Bloom Filter的层数来完成，每一层容量将增长，但是随着不断的扩容，TairBloom内部的层数会越来越多，此时会导致完成查询任务需要遍历多层Bloom Filter，性能将严重下降。
  - `BF.RESERVE`（或 `BF.INSERT`）：执行时需要设置capacity（初始容量），该命令会在TairBloom的第一层初始化设置的容量，在TairBloom内部的Bloom Filter层数少，查询速度快。

> **说明** 以插入10,000,000个元素、错误率为0.01为例，直接通过 `BF.ADD` 创建，TairBloom需占用176 MB；而通过 `BF.RESERVE` 创建时仅占用16 MB。

虽然TairBloom支持扩容，但在实际使用过程中请避免发生扩容操作，建议将该功能视为保障措施，若实际容量超过预设容量时，TairBloom能通过扩容操作，保障业务正常写入，规避线上事故。
下表为通过 `BF.RESERVE` 创建不同初始容量和错误率的key所占用的内存，仅供参考。

容量（元素的个数）	false positive:0.01	false positive:0.001	false positive:0.0001
100,000	0.12 MB	0.25 MB	0.25 MB
1,000,000	2 MB	2 MB	4 MB
10,000,000	16 MB	32 MB	32 MB
100,000,000	128 MB	256 MB	256 MB
1,000,000,000	2 GB	2 GB	4 GB

- 由于TairBloom只能插入新元素且无法删除已有元素，因此TairBloom的内存占用量只会增加不会减少。为防止TairBloom越来越大，甚至导致Redis内存溢出（Out Of Memory），向您提供如下使用建议。
  - 拆分业务数据：拆分、细化业务数据，避免将大量数据存入一个TairBloom中，这样不仅会导致这个key过大，影响查询性能，同时也会由于这个key中插入了过多数据，大部分的查询流量都会请求到这个key所在的Redis实例上，从而造成热点key，甚至发生访问倾斜的情况。
    请拆分业务数据，将数据分散到多个TairBloom中，若您的实例为集群实例，您可以将TairBloom分散到集群内的各个实例上，均衡内存容量与流量，充分发挥分布式集群的优势。
  - 定期重建：如果业务允许，您可以定期重建TairBloom，通过 `DEL` 删除TairBloom，从后端数据库拉取数据并重建TairBloom，以控制TairBloom的大小。
    您也可以在初期创建多个TairBloom，并采用多个TairBloom轮转切换的方式实现控制单个TairBloom的大小。该方案的优点为仅需创建一次TairBloom，无需频繁地重建TairBloom，缺点是需要创建多个TairBloom，会浪费部分内存空间。

命令列表

TairBloom命令

命令	语法	说明
BF.RESERVE	`BF.RESERVE key error_rate capacity`	创建一个大小为capacity，错误率为error_rate的空的TairBloom。
BF.ADD	`BF.ADD key item`	在key指定的TairBloom中添加一个元素。
BF.MADD	`BF.MADD key item [item ...]`	在key指定的TairBloom中添加多个元素。

命令	语法	说明
BF.EXISTS	`BF.EXISTS key item`	检查一个元素是否存在于key指定的TairBloom中。
BF.MEXISTS	`BF.MEXISTS key item [item ...]`	同时检查多个元素是否存在于key指定的TairBloom中。
BF.INSERT	`BF.INSERT key [CAPACITY cap] [ERROR error] [NOCREATE] ITEMS item [item ...]`	在key指定的TairBloom中一次性添加多个元素，添加时可以指定大小和错误率，且可以控制在TairBloom不存在的时候是否自动创建。
DEL	`DEL key [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairBloom数据。 **说明** 已加入TairBloom数据中的元素无法单独删除，您可以使用DEL命令删除整条TairBloom数据。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

BF.RESERVE

类别	说明
语法	`BF.RESERVE key error_rate capacity`
时间复杂度	O(1)
命令描述	创建一个大小为capacity，错误率为error_rate的空的TairBloom。
选项	• Key：Key名称（TairBloom数据结构），用于指定作为命令调用对象的TairBloom。 • error_rate：期望的错误率（False Positive Rate），该值必须介于0和1之间。该值越小，精度越高，TairBloom的内存占用量越大，CPU使用率越高。 • capacity：TairBloom的初始容量，即期望添加到TairBloom中的元素的个数。 当实际添加的元素个数超过该值时，TairBloom将通过增加Bloom Filter的层数完成自动扩容，该过程会导致查询性能下降。每增加一层，就可能需要遍历多层Bloom Filter来完成查询。因此，如果对性能非常的敏感，需要在使用前充分评估要添加到TairBloom的元素个数，避免发生导致层数增加的扩容操作。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`BF.RESERVE BFKEY 0.01 100`
	返回示例：
	`OK`

BF.ADD

类别	说明
语法	`BF.ADD key item`
时间复杂度	O(log N)，其中N是TairBloom的层数。
命令描述	在key指定的TairBloom中添加一个元素。 说明 若目标key不存在，Tair会自动创建一个TairBloom，创建TairBloom的默认容量（capacity）为100，错误率（error_rate）为0.01。
选项	• Key：Key名称（TairBloom数据结构），用于指定作为命令调用对象的TairBloom。 • item：需要添加到TairBloom的元素。
返回值	• 1：表示该元素之前一定不存在，并往TairBloom中添加该元素。 • 0：表示该元素可能已存在，所以不会进行添加或更新操作。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`BF.ADD BFKEY item1`
	返回示例：
	`(integer) 1`

BF.MADD

类别	说明
语法	`BF.MADD key item [item ...]`
时间复杂度	O(log N)，其中N是TairBloom的层数。
命令描述	在key指定的TairBloom中添加多个元素。 说明 若目标key不存在，Tair会自动创建一个TairBloom，创建TairBloom的默认容量（capacity）为100，错误率（error_rate）为0.01。
选项	• Key：Key名称（TairBloom数据结构），用于指定作为命令调用对象的TairBloom。 • item：需要添加到TairBloom的元素，可设置多个。
返回值	• 1：表示该元素之前一定不存在，并往TairBloom中添加该元素。 • 0：表示该元素可能已存在，所以不会进行添加或更新操作。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`BF.MADD BFKEY item1 item2 item3`
	返回示例：
	`(integer) 1`
	`(integer) 1`
	`(integer) 1`

BF.EXISTS

类别	说明
语法	`BF.EXISTS key item`
时间复杂度	O(log N)，其中N是TairBloom的层数。
命令描述	检查一个元素是否存在于key指定的TairBloom中。
选项	• Key：Key名称（TairBloom数据结构），用于指定作为命令调用对象的TairBloom。 • item：需要查询的元素。
返回值	• 0：表示该元素一定不存在。 • 1：表示该元素可能存在。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`BF.EXISTS BFKEY item1`
	返回示例：
	`(integer) 1`

BF.MEXISTS

类别	说明
语法	`BF.MEXISTS key item [item ...]`
时间复杂度	O(log N)，其中N是TairBloom的层数。
命令描述	同时检查多个元素是否存在于key指定的TairBloom中。
选项	• Key：Key名称（TairBloom数据结构），用于指定作为命令调用对象的TairBloom。 • item：需要查询的元素，可设置多个。
返回值	• 0：表示该元素一定不存在。 • 1：表示该元素可能存在。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`BF.MEXISTS BFKEY item1 item5`
	返回示例：
	`(integer) 1`
	`(integer) 0`

BF.INSERT

类别	说明
语法	`BF.INSERT key [CAPACITY cap] [ERROR error] [NOCREATE] ITEMS item [item ...]`

类别	说明
时间复杂度	O(log N)，其中N是TairBloom的层数。
命令描述	在key指定的TairBloom中一次性添加多个元素，添加时可以指定大小和错误率，且可以控制在TairBloom不存在的时候是否自动创建。
选项	• Key：Key名称（TairBloom数据结构），用于指定作为命令调用对象的TairBloom。 • capacity：TairBloom的初始容量，即期望添加到TairBloom中的元素的个数，当TairBloom已经存在时该值将被忽略。 当实际添加的元素个数超过该值时，TairBloom将进行自动的扩容，该过程会导致性能有所下降，下降的程度是随着元素个数的增长而下降的，这是因为TairBloom的扩容是通过增加Bloom Filter的层数来完成的。每增加一层，在查询的时候就可能会遍历多层Bloom Filter来完成，每一层的容量都是上一层的两倍。因此，如果对性能非常的敏感，需要在使用前充分评估要添加到TairBloom的元素个数，避免发生扩容操作。 • error_rate：期望的错误率（False Positive Rate），该值必须介于0和1之间。该值越小，精度越高，TairBloom的内存占用量越大，CPU使用率越高。 • NOCREATE：设置该选项后，当指定的TairBloom不存在的时候不要自动创建该TairBloom。该参数不能与CAPACITY和ERROR同时设置。 • item：需要添加的元素，可设置多个。
返回值	• 1：表示该元素之前一定不存在，并往TairBloom中添加该元素。 • 0：表示该元素可能已存在，所以不会进行添加或更新操作。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`BF.INSERT bfkey1 CAPACITY 10000 ERROR 0.001 ITEMS item1 item2 item3`
	返回示例：
	`(integer) 1`
	`(integer) 1`
	`(integer) 1`

3.6.7. TairDoc

TairDoc是一种完全兼容RedisJSON的文档数据结构，支持JSON数据的增删改查。

主要特性

- 完整地支持JSON标准。
- 接口兼容RedisJSON。
- 支持JSONPointer和JSONPath两种语法。
- 文档作为二进制树存储，可以快速访问JSON数据的子元素。
- 支持JSON到XML或YAML格式的转换。

前提条件

实例为Tair（Redis企业版）性能增强型。

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

操作对象为性能增强型实例中的TairDoc数据。

命令列表

TairDoc命令

命令	语法	说明
JSON.SET	`JSON.SET key path json [NX | XX]`	创建key并将JSON的值存储在对应的path中，若key及目标path已经存在，则更新对应的JSON值。
JSON.GET	`JSON.GET key path [FORMAT XML | YAML] [ROOTNAME root] [ARRNAME arr]`	获取目标key、path中存储的JSON数据。
JSON.DEL	`JSON.DEL key path`	删除目标key中path对应的JSON数据，若未指定path，则删除key。若指定的key不存在或path不存在，则忽略。
JSON.TYPE	`JSON.TYPE key path`	获取目标key中path对应值的类型，结果可能包括 `boolean` 、 `string` 、 `number` 、 `array` 、 `object` 、 `raw` 、 `reference` 、 `const` 、 `null` 等。
JSON.NUMINCRBY	`JSON.NUMINCRBY key path value`	对目标key中path对应的值增加value，path对应的值和待增加的value必须是int或double类型。
JSON.STRAPPEND	`JSON.STRAPPEND key path json-string`	在指定path对应值中添加json-string字符串，path对应值的类型也需要为字符串。
JSON.STRLEN	`JSON.STRLEN key path`	获取目标key中path对应值的字符串长度，path对应值的类型需要为字符串。
JSON.ARRAPPEND	`JSON.ARRAPPEND key path json [json ...]`	在指定path对应数组（array）的末尾添加JSON数据，支持添加多个JSON。
JSON.ARRPOP	`JSON.ARRPOP key path [index]`	移除并返回path对应数组（array）中指定位置（index）的元素。
JSON.ARRINSERT	`JSON.ARRINSERT key path [index] json [json ...]`	将JSON插入到path对应的数组（array）中，原有元素会往后移动。
JSON.ARRLEN	`JSON.ARRLEN key path`	获取path对应数组（array）的长度。
JSON.ARRTRIM	`JSON.ARRTRIM key path start stop`	修剪目标key的path对应的数组（array），保留start至stop范围内的数据。
DEL	`DEL <key> [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairDoc数据。

说明 本文的命令语法定义如下：

- `大写关键字` ：命令关键字。
- *斜体* ：变量。
- `[options]` ：可选参数，不在括号中的参数为必选。
- `A|B` ：该组参数互斥，请进行二选一或多选一。
- `...` ：前面的内容可重复。

JSON.SET

类别	说明
语法	`JSON.SET key path json [NX | XX]`
时间复杂度	O(N)

类别	说明
命令描述	创建key并将JSON的值存储在对应的path中，若key及目标path已经存在，则更新对应的JSON值。 **说明** 若key不存在，path必须是root（即 `.` ）。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • json：待新增或更新的JSON数据。 • NX：当path不存在时写入。 • XX：当path存在时写入。
返回值	• 执行成功：OK。 • 指定了XX且path不存在：nil。 • 指定了NX且path已存在：nil。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`JSON.SET doc . '{"foo": "bar", "baz" : 42}'`
	返回示例：
	`OK`

JSON.GET

类别	说明
语法	`JSON.GET key path [FORMAT XML | YAML] [ROOTNAME root] [ARRNAME arr]`
时间复杂度	O(N)
命令描述	获取目标key、path中存储的JSON数据。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • FORMAT：指定返回的JSON格式，支持XML、YAML格式。 • ROOTNAME：指定XML语法ROOT元素的标签。 • ARRNAME：指定XML语法ARRAY元素的标签。 **说明** ROOTNAME与ARRNAME参数需在指定FORMAT参数为XML时配合使用。
返回值	• 执行成功：对应的JSON数据。 • 其它情况返回相应的异常信息。

类别	说明

示例	提前执行 `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` 命令。
	命令示例：
	`JSON.GET doc . FORMAT XML ROOTNAME ROOT ARRNAME ARR`
	返回示例：
	`"<?xml version=\"1.0\" encoding=\"UTF-8\"?><ROOT><foo>bar</foo><baz>42</baz></ROOT>"`

JSON.DEL

类别	说明
语法	`JSON.DEL key path`
时间复杂度	O(N)
命令描述	删除目标key中path对应的JSON数据，若未指定path，则删除key。若指定的key不存在或path不存在，则忽略。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。
返回值	• 执行成功：1。 • 执行失败：0。 • 其它情况返回相应的异常信息。

示例	提前执行 `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` 命令。
	命令示例：
	`JSON.DEL doc .foo`
	返回示例：
	`(integer) 1`

JSON.TYPE

类别	说明
语法	`JSON.TYPE key path`
时间复杂度	O(N)
命令描述	获取目标key中path对应值的类型，结果可能包括 `boolean`、`string`、`number`、`array`、`object`、`raw`、`reference`、`const`、`null` 等。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。

类别	说明
返回值	• 执行成功：返回查询到的类型。 • 执行失败：0。 • 若key或path不存在：nil。 • 其它情况返回相应的异常信息。

示例	提前执行 `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` 命令。
	命令示例：
	`JSON.TYPE doc .foo`
	返回示例：
	`string`

JSON.NUMINCRBY

类别	说明
语法	`JSON.NUMINCRBY key path value`
时间复杂度	O(N)
命令描述	对目标key中path对应的值增加value，path对应的值和待增加的value必须是int或double类型。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • value：待增加的数值。
返回值	• 执行成功：返回操作完成后path对应的值。 • 若key或path不存在：error。 • 其它情况返回相应的异常信息。

示例	提前执行 `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` 命令。
	命令示例：
	`JSON.NUMINCRBY doc .baz 10`
	返回示例：
	`"52"`

JSON.STRAPPEND

类别	说明
语法	`JSON.STRAPPEND key path json-string`
时间复杂度	O(N)
命令描述	在指定path对应值中添加json-string字符串，path对应值的类型也需要为字符串。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • json-string：待添加到path对应值的字符串。

类别	说明
返回值	• 执行成功：返回操作完成后path对应值的字符串长度。 • key不存在：-1。 • 其它情况返回相应的异常信息。

示例	提前执行 `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` 命令。
	命令示例：
	`JSON.STRAPPEND doc .foo rrrrr`
	返回示例：
	`(integer) 8`

JSON.STRLEN

类别	说明
语法	`JSON.STRLEN key path`
时间复杂度	O(N)
命令描述	获取目标key中path对应值的字符串长度，path对应值的类型需要为字符串。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。
返回值	• 执行成功：返回path对应值的字符串长度。 • key不存在：-1。 • 其它情况返回相应的异常信息。

示例	提前执行 `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` 命令。
	命令示例：
	`JSON.STRLEN doc .foo`
	返回示例：
	`(integer) 3`

JSON.ARRAPPEND

类别	说明
语法	`JSON.ARRAPPEND key path json [json ...]`
时间复杂度	O(M*N)，M是需要插入的元素（json）数量，N是数组元素数量。
命令描述	在指定path对应数组（array）的末尾添加JSON数据，支持添加多个JSON。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • json：需要插入的数据。

类别	说明
返回值	• 执行成功：返回操作完成后数组（array）中的元素数量。 • key不存在：-1。 • 其它情况返回相应的异常信息。

示例	提前执行 `JSON.SET doc . '{"id": [1,2,3]}'` 命令。
	命令示例：
	`JSON.ARRAPPEND doc .id null false true`
	返回示例：
	`(integer) 6`

JSON.ARRPOP

类别	说明
语法	`JSON.ARRPOP key path [index]`
时间复杂度	O(M*N)，M是key包含的子元素，N是数组元素数量。
命令描述	移除并返回path对应数组（array）中指定位置（index）的元素。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • index：数组的索引，起始下标为0，负数表示反向取值，若不传该参数默认为最后一个元素。
返回值	• 执行成功：移除并返回该元素。 • 数组为空数组：‘ERR array index outflow’。 • 其它情况返回相应的异常信息。

示例	提前执行 `JSON.SET doc . '{"id": [1,2,3]}'` 命令。
	命令示例：
	`JSON.ARRPOP doc .id 0`
	返回示例：
	`"1"`

JSON.ARRINSERT

类别	说明
语法	`JSON.ARRINSERT key path [index] json [json ...]`
时间复杂度	O(M*N)，M是要插入的元素（json）数量，N是数组元素数量。
命令描述	将JSON插入到path对应的数组（array）中，原有元素会往后移动。

类别	说明
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • index：数组的索引，起始下标为0，负数表示反向取值，若不传该参数默认为最后一个元素。 • json：需要插入的数据。
返回值	• 执行成功：返回操作完成后数组（array）中的元素数量。 • 数组为空数组：‘ERR array index outflow’。 • 其它情况返回相应的异常信息。

示例	提前执行`JSON.SET doc . '{"id": [1,2,3]}'`命令。
	命令示例：
	`JSON.ARRINSERT doc .id 0 10 15`
	返回示例：
	`(integer) 5`

JSON.ARRLEN

类别	说明
语法	`JSON.ARRLEN key path`
时间复杂度	O(N)
命令描述	获取path对应数组（array）的长度。
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。
返回值	• 执行成功：数组（array）的长度。 • key不存在：-1。 • 其它情况返回相应的异常信息。

示例	提前执行`JSON.SET doc . '{"id": [1,2,3]}'`命令。
	命令示例：
	`JSON.ARRLEN doc .id`
	返回示例：
	`(integer) 3`

JSON.ARRTRIM

类别	说明
语法	`JSON.ARRTRIM key path start stop`
时间复杂度	O(N)
命令描述	修剪目标key的path对应的数组（array），保留start至stop范围内的数据。

类别	说明
选项	• key：TairDoc的key，用于指定作为命令调用对象的TairDoc。 • path：目标key的path。 • start：修剪的开始位置，取值为从0开始的一个索引值，修剪后的数组包含该位置的元素。 • stop：修剪的结束位置，取值为从0开始的一个索引值，修剪后的数组包含该位置的元素。
返回值	• 执行成功：返回操作完成后数组的长度。 • key不存在：-1。 • 其它情况返回相应的异常信息。

示例	提前执行`JSON.SET doc . '{"id": [1,2,3,4,5,6]}'`命令。
	命令示例：
	`JSON.ARRTRIM doc .id 3 4`
	返回示例：
	`(integer) 2`

JSONPointer和JSONPath

TairDoc在支持JSONPointer的基础上支持了JSONPath的部分语法，示例如下。

提前执行`JSON.SET doc . '{"foo": "bar", "baz" : [1,2,3]}'`命令。

JSONPointer	JSONPath

命令示例：
`JSON.GET doc .baz[0]`
返回示例：
`"1"`
	命令示例：
	`JSON.GET doc /baz/0`
	返回示例：
	`"1"`

具体的兼容方式如下表所示。

兼容项	JSONPath	JSONPointer
根元素	.	“”
获取元素	.a.b.c	/a/b/c
数组	.a[2]	/a/2
整体获取	.a["b.c"]	/a/b.c
	.a['b.c']	/a/b.c

3.6.8. TairTS

TairTS是基于Redis Module开发的时序数据结构，提供低时延、高并发的内存读写访问，及快速地过滤、聚合查询功能，集存储与计算为一体，在简化了处理时序数据流程的同时，大幅度提高了性能。

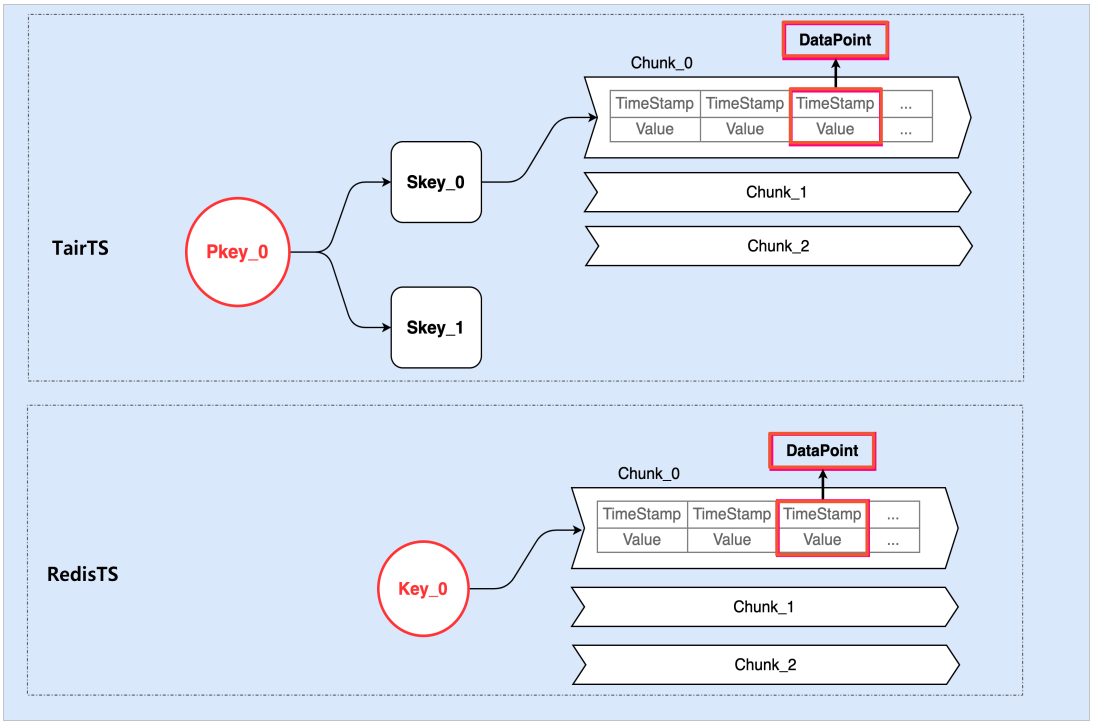
TairTS简介

相比RedisTimeSeries，TairTS提供了更丰富的功能：

- 通过独创的Pkey（额外一层Hash结构），轻松实现Pkey级别（多时间线）聚合查询。

例如您可以在foo（Pkey）中创建以各项指标名称与设备ID命名的Skey，例如temperature:1、pressure:1、distance:1等，可通过TairTS自带的EXTS.S.MRANGE命令轻松获取设备ID为1的自定义监控信息，而使用RedisTimeSeries则需要在业务逻辑代码中嵌入大量数据聚合运算才能实现该功能。

TairTS与RedisTS数据结构对比图



TairTS数据结构解析：

- Pkey（一组时间线）：TairTS数据结构，可包含多个Skey。
- Skey（一条时间线）：可包含多个固定容量的Chunk，每个Skey可设置不同的Label（标签），可通过Label在海量数据中快速过滤目标Skey。
- Chunk（数据块）：可存储多个DataPoint。
  - Chunk的容量支持自定义，最多包含256个DataPoint。
  - Chunk为最小的过期单元，即单个Chunk中所有Datapoint都过期后才会删除该Chunk。
- DataPoint（时序数据）：包含一个时间戳和一个value数据（double类型）。

- 支持降采样、属性过滤、分批查询、多种数值函数等条件下的聚合操作，将批量查询与聚合计算集成到单条命令中，减少网络交互，实现毫秒级响应。
- 支持对历史时序数据的更新或累加。
- 支持时间线级别的TTL设定，保证每条时间线都可以按时间窗口自动滚动。
- 采用高效的Gorilla压缩算法与特定存储，极大降低存储成本。

典型场景

- 监控数据的存储与计算
- APM秒级监控
- IoT（物联网）数据分析与处理
- 限流风控
- 热点消息的缓存
- 时间窗口函数

最佳实践

基于TairTS实现秒级监控

前提条件

实例为Tair（Redis企业版）性能增强型，且小版本为1.7.20及以上。

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

- 操作对象为性能增强型实例中的TairTS数据。
- TairTS的优势为实时、高并发的写入与查询性能，缺陷为存储容量有限，请合理设置TTL，及时淘汰过期数据。
- 为节省内存空间，关于设置 `CHUNK_SIZE` 的建议如下：
  - 若Skey（时间线）的平均数据点大于5,000个，设置 `CHUNK_SIZE` 为256（默认值）。
  - 若Skey（时间线）的平均数据点小于5,000个，设置 `CHUNK_SIZE` 为平均数据点个数 / 20。例如某Skey的平均数据点为1,000，可以设置 `CHUNK_SIZE` 为50。

命令列表

TairTS命令

类型	命令	语法	说明
基础写操作	EXTS.P.CREATE	`EXTS.P.CREATE Pkey`	创建一个新的Pkey（TairTS数据结构），若Pkey已存在则创建失败。
	EXTS.S.CREATE	`EXTS.S.CREATE Pkey Skey [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 label2 val2 ...]`	在指定的Pkey中创建一个Skey，若Pkey不存在则会自动创建，若Skey已经存在则创建失败。 **说明** 您可以在创建Skey时设置其相关属性，例如过期时间、是否开启压缩等。
	EXTS.S.ALTER	`EXTS.S.ALTER Pkey Skey [DATA_ET time]`	修改指定Skey的元数据信息，当前仅支持修改过期时间（DATA_ET）。
	EXTS.S.ADD	`EXTS.S.ADD Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	向Skey中插入一条Datapoint数据。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
	EXTS.S.MADD	`EXTS.S.MADD Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	在指定Pkey的多个Skey分中别插入一条Datapoint数据。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
	EXTS.S.INCRBY	`EXTS.S.INCRBY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	向Skey中插入一条Datapoint数据，该命令中的value将与Skey中最近Datapoint的value值相加实现递增，也可以指定该命令中的value为负数实现递减。若Pkey或Skey不存在则会自动创建，默认初始值为0，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
	EXTS.S.MINCRBY	`EXTS.S.MINCRBY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	在指定Pkey的多个Skey分别插入一条Datapoint数据，该命令中的value将分别与各个Skey中最近Datapoint的value值相加实现递增，也可以指定该命令中的value为负数实现相减。若Pkey或Skey不存在则会自动创建，默认初始值为0，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
	EXTS.S.DEL	`EXTS.S.DEL Pkey Skey`	删除指定Pkey中的单个Skey，并删除目标Skey中所有的Datapoint数据。
	EXTS.S.GET	`EXTS.S.GET Pkey Skey`	查询指定Skey中最新的Datapoint数据。

类型	命令	语法	说明
基础读操作	EXTS.S.INFO	`EXTS.S.INFO Pkey Skey`	查询指定Skey的元数据信息，包含Datapoint数量、最近Datapoint的时间戳与value值、Skey的标签信息等信息。
	EXTS.S.QUERYINDEX	`EXTS.S.QUERYINDEX Pkey filter1 [filter2 ...]`	在Pkey中自定义过滤条件（filter），查询目标Skey。
聚合操作	EXTS.S.RANGE	`EXTS.S.RANGE Pkey Skey fromTs toTs [MAXCOUNT count] [AGGREGATION aggregationType timeBucket]`	在Skey中查询指定时间内（包含指定时间点）的Datapoint数据。
	EXTS.S.MRANGE	`EXTS.S.MRANGE Pkey fromTs toTs [MAXCOUNT count] [AGGREGATION aggregationType timeBucket] [WITHLABELS] FILTER filter1 [filter2 ...]`	在Skey中自定义过滤条件（filter）与查询时间点（包含指定时间点），查询目标Datapoint数据。
	EXTS.P.RANGE	`EXTS.P.RANGE Pkey fromTs toTs pkeyAggregationType pkeyTimeBucket [MAXCOUNT count] [AGGREGATION aggregationType timeBucket] [WITHLABELS] FILTER filter1 [filter2 ...]`	在Pkey层级对符合过滤条件（filter）的Datapoint数据进行聚合，若您指定了Skey层级的聚合，则会优先进行Skey层级聚合（效果与EXTS.S.MRANGE命令相同），再从Pkey层级对第一次聚合结果进行二次聚合。
并发写操作	EXTS.S.RAW_MODIFY	`EXTS.S.RAW_MODIFY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	修改指定Skey中Datapoint数据的value值。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
	EXTS.S.RAW_MMODIFY	`EXTS.S.RAW_MMODIFY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	批量修改多个指定Skey中Datapoint数据的value值。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
	EXTS.S.RAW_INCRBY	`EXTS.S.RAW_INCRBY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	该命令中的value值会与指定Skey中Datapoint数据的value相加实现递增，也可以指定该命令中的value为负数实现递减。若Pkey或Skey不存在则会自动创建，默认初始值为0，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
	EXTS.S.RAW_MINCRBY	`EXTS.S.RAW_MINCRBY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	批量修改多个指定Skey中Datapoint数据的value值，该命令中的value值会与指定Skey中Datapoint数据的value相加实现递增，也可以指定该命令中的value为负数实现递减。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
通用	DEL	`DEL key [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairTS数据。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

EXTS.P.CREATE

类别	说明
语法	`EXTS.P.CREATE Pkey`
时间复杂度	O(1)
命令描述	创建一个新的Pkey（TairTS数据结构），若Pkey已存在则创建失败。
选项	• Pkey：Key名称（TairTS数据结构），用于指定命令调用的TairTS对象。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.P.CREATE foo`
	返回示例：
	`OK`

EXTS.S.CREATE

类别	说明
语法	`EXTS.S.CREATE Pkey Skey [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 label2 val2 ...]`
时间复杂度	O(1)
命令描述	在指定的Pkey中创建一个Skey，若Pkey不存在则会自动创建，若Skey已经存在则创建失败。 **说明** 您可以在创建Skey时设置其相关属性，例如过期时间、是否开启压缩等。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.CREATE foo temperature DATA_ET 10000000 LABELS sensor_id 1`
	返回示例：
	`OK`

EXTS.S.ALTER

类别	说明
语法	`EXTS.S.ALTER Pkey Skey [DATA_ET time]`
时间复杂度	O(1)
命令描述	修改指定Skey的元数据信息，当前仅支持修改过期时间（DATA_ET）。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.ALTER foo temperature DATA_ET 100000`
	返回示例：
	`OK`

EXTS.S.ADD

类别	说明
语法	`EXTS.S.ADD Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
时间复杂度	O(1)
命令描述	向Skey中插入一条Datapoint数据。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。 • ts：Datapoint数据的Unix时间戳，单位为毫秒，支持用 `*` 表示系统当前的时间戳。 • value：Datapoint数据的值，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.ADD foo temperature 1644310456023 30.5 DATA_ET 1000000 LABELS sensor_id 1`
	返回示例：
	`OK`

EXTS.S.MADD

类别	说明
语法	`EXTS.S.MADD Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
时间复杂度	O(n)，其中n为keynumber。
命令描述	在指定Pkey的多个Skey分中别插入一条Datapoint数据。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • keynumber：指定多条数据的个数。 • Skey：Skey名称。 • ts：Datapoint数据的Unix时间戳，单位为毫秒，支持用 `*` 表示系统当前的时间戳。 • value：Datapoint数据的值，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.MADD foo 3 temperature * 30.2 pressure * 2.05 distance * 0.5`
	返回示例：
	`1) OK`
	`2) OK`
	`3) OK`

EXTS.S.INCRBY

类别	说明
语法	`EXTS.S.INCRBY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
时间复杂度	O(1)
命令描述	向Skey中插入一条Datapoint数据，该命令中的value将与Skey中最近Datapoint的value值相加实现递增，也可以指定该命令中的value为负数实现递减。若Pkey或Skey不存在则会自动创建，默认初始值为0，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。 • ts：Datapoint数据的Unix时间戳，单位为毫秒，支持用 `*` 表示系统当前的时间戳。 • value：待增加操作的值，可以指定该值为负数实现相减，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。

类别	说明
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXTS.S.ADD foo temperature 1644310456023 30.0` 命令。
	命令示例：
	`EXTS.S.INCRBY foo temperature 1644372093031 2`
	
	返回示例：
	`OK`
	
	若此时执行 `EXTS.S.GET foo temperature` 命令，将会返回如下结果：
	`1) (integer) 1644372093031`
	`2) "32"`

EXTS.S.MINCRBY

类别	说明
语法	`EXTS.S.MINCRBY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
时间复杂度	O(n)，其中n为keynumber。
命令描述	在指定Pkey的多个Skey分别插入一条Datapoint数据，该命令中的value将分别与各个Skey中最近Datapoint的value值相加实现递增，也可以指定该命令中的value为负数实现相减。若Pkey或Skey不存在则会自动创建，默认初始值为0，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • keynumber：指定多条数据的个数。 • Skey：Skey名称。 • ts：Datapoint数据的Unix时间戳，单位为毫秒，支持用 `*` 表示系统当前的时间戳。 • value：待增加操作的值，可以指定该值为负数实现相减，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.MINCRBY foo 3 temperature * 0.2 pressure * -0.1 distance * 0.0`
	
	返回示例：
	`1) OK`
	`2) OK`
	`3) OK`

EXTS.S.DEL

类别	说明
语法	`EXTS.S.DEL Pkey Skey`
时间复杂度	O(1)
命令描述	删除指定Pkey中的单个Skey，并删除目标Skey中所有的Datapoint数据。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.DEL foo temperature`
	返回示例：
	`OK`

EXTS.S.GET

类别	说明
语法	`EXTS.S.GET Pkey Skey`
时间复杂度	O(1)
命令描述	查询指定Skey中最新的Datapoint数据。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。
返回值	• 执行成功：返回对应的Datapoint数据。 • nil：表示Pkey或Skey不存在。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.GET foo temperature`
	返回示例：
	`1) (integer) 1644372730150`
	`2) "32.2"`

EXTS.S.INFO

类别	说明
语法	`EXTS.S.INFO Pkey Skey`
时间复杂度	O(1)
命令描述	查询指定Skey的元数据信息，包含Datapoint数量、最近Datapoint的时间戳与value值、Skey的标签信息等信息。

类别	说明
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。
返回值	• 执行成功：返回Skey的元数据信息。 • nil：表示Pkey或Skey不存在。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.INFO foo temperature`
	返回示例：
	`1) totalDataPoints // Datapoint数量。`
	`2) (integer) 1`
	`3) maxDataPoints // Skey可存储Datapoint数量的上限，默认为0（表示不限制）。`
	`4) (integer) 0`
	`5) maxDataPointsPerChunk // 每个Chunk存储的Datapoint个数。`
	`6) (integer) 32`
	`7) dataPointsExpireTime // Skey的相对过期时间（DATA_ET），单位为毫秒，0表示不过期。`
	`8) (integer) 0`
	`9) lastTimestamp // 最近Datapoint的时间戳。`
	`10) (integer) 1644389400996`
	`11) chunkCount // Skey的chunk数量。`
	`12) (integer) 1`
	`13) lastValue // 最近Datapoint的value。`
	`14) (integer) 28`
	`15) labels // Skey的标签信息。`
	`16) 1) 1) "sensor_id"`
	`2) "1"`

EXTS.S.QUERYINDEX

类别	说明
语法	`EXTS.S.QUERYINDEX Pkey filter1 [filter2 ...]`
时间复杂度	O(n)，其中n为过滤条件中的最大集合数。
命令描述	在Pkey中自定义过滤条件（filter），查询目标Skey。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • filter：过滤条件，您可以根据Skey的标签（LABELS）过滤目标Skey，更多信息请参见索引过滤语法。 **说明** 构建filter时，必须存在EQ、CONTAINS、LIST_MATCH逻辑中的任意一个，否则会查询失败。
返回值	• 执行成功：返回符合过滤条件的Skey。 • nil：表示Pkey或Skey不存在。 • 其它情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`EXTS.S.QUERYINDEX foo sensor_id=1`
	返回示例：
	`1) "temperature"`

EXTS.S.RANGE

类别	说明
语法	`EXTS.S.RANGE` *`Pkey Skey fromTs toTs`* `[MAXCOUNT` *`count`*`] [AGGREGATION` *`aggregationType timeBucket`*`]`
时间复杂度	O(n)，其中n为目标Datapoint的数据块个数。
命令描述	在Skey中查询指定时间内（包含指定时间点）的Datapoint数据。

选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。
	• Skey：Skey名称。
	• fromTs：查询的开始时间（Unix时间戳），单位为毫秒。
	• toTs：查询的结束时间（Unix时间戳），单位为毫秒，支持用 `*` 表示系统当前的时间戳，若该值等于fromTs可实现单时间点查询。
	• MAXCOUNT：指定返回的Datapoint条数，默认为不填（Tair的上限为1,000,000条）。
	• AGGREGATION：
	  ◦ aggregationType：聚合类型，例如 `MAX`（最大值）、`AVG`（平均值）、`SUM`（求和）等，更多信息请参见聚合功能语法。
	  ◦ timeBucket：采样间隔，单位为毫秒，最小值为1,000毫秒。
	Tair会将该时间范围内的数据进行聚合并返回一个结果，返回的时间点为采样间隔的开始时间。
	例如 `AGGREGATION AVG 5000` 将返回每5,000ms的平均数。

返回值	• 执行成功：返回对应的Datapoint数据，若命令中指定了聚合，则返回聚合结果。 **说明** 返回结果中会额外返回一个token值，0表示已全部显示，1表示还有符合条件的Datapoint数据未显示。您可以根据该值，同时将已返回结果中最后一个Datapoint的时间戳作为开始时间继续遍历获取，轻松实现分批聚合。 • nil：表示Pkey或Skey不存在。 • 其它情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`EXTS.S.RANGE foo test 1644459031662 * AGGREGATION AVG 10000 MAXCOUNT 2`
	`// 求指定时间点内每10,000ms的平均数，同时指定返回2条数据。`
	
	返回示例：
	`1) 1) 1) (integer) 1644459730000`
	`2) "20.6"`
	`2) 1) (integer) 1644459790000`
	`2) "21.2"`
	`2) (integer) 1 // 1表示还有符合条件的Datapoint数据未显示，0表示已全部显示。`

EXTS.S.MRANGE

类别	说明
语法	`EXTS.S.MRANGE Pkey fromTs toTs [MAXCOUNT count] [AGGREGATION aggregationType timeBucket] [WITHLABELS] FILTER filter1 [filter2 ...]`
时间复杂度	O(n)，其中n为目标Datapoint的数据块个数。
命令描述	在Skey中自定义过滤条件（filter）与查询时间点（包含指定时间点），查询目标Datapoint数据。

选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。
	• fromTs：查询的开始时间（Unix时间戳），单位为毫秒。
	• toTs：查询的结束时间（Unix时间戳），单位为毫秒，支持用 `*` 表示系统当前的时间戳，若该值等于fromTs可实现单时间点查询。
	• MAXCOUNT：指定每个Skey返回的Datapoint条数，默认为不填（Tair的上限为1,000,000条）。
	• AGGREGATION：
	  ◦ aggregationType：聚合类型，例如 `MAX`（最大值）、`AVG`（平均值）、`SUM`（求和）等，更多信息请参见聚合功能语法。
	  ◦ timeBucket：采样间隔，单位为毫秒，最小值为1,000毫秒。
	Tair会将该时间范围内的数据进行聚合并返回一个结果，返回的时间点为采样间隔的开始时间。
	• WITHLABELS：设置返回结果中是否包含标签信息，默认为不填（不显示标签信息）。
	• filter：过滤条件，您可以根据Skey的标签（LABELS）过滤目标Skey，更多信息请参见索引过滤语法。
	
	说明 构建filter时，必须存在EQ、CONTAINS、LIST_MATCH逻辑中的任意一个，否则会查询失败。

返回值	• 执行成功：返回符合过滤条件的Skey组信息。 • nil：表示Pkey或Skey不存在。 • 其它情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`EXTS.S.MRANGE foo 1644451031662 * AGGREGATION MAX 10000 WITHLABELS FILTER sensor_id=1`
	
	返回示例：
	`1) 1) "temperature"`
	`   2) 1) 1) "sensor_id"`
	`         2) "1"`
	`   3) 1) 1) (integer) 1644481000000`
	`         2) "30"`
	`   4) (integer) 0`
	`2) 1) "test"`
	`   2) 1) 1) "sensor_id"`
	`         2) "1"`
	`   3) 1) 1) (integer) 1644459730000`
	`         2) "20"`
	`      2) 1) (integer) 1644459790000`
	`         2) "20"`
	`      3) 1) (integer) 1644460620000`
	`         2) "29"`
	`   4) (integer) 0`

EXTS.P.RANGE

类别	说明
语法	`EXTS.P.RANGE` *Pkey fromTs toTs pkeyAggregationType pkeyTimeBucket* `[MAXCOUNT` *count*`]` `[AGGREGATION` *aggregationType timeBucket*`]` `[WITHLABELS] FILTER` *filter1* `[`*filter2* `...]`
时间复杂度	O(n)，其中n为目标Datapoint的数据块个数。
命令描述	在Pkey层级对符合过滤条件（filter）的Datapoint数据进行聚合，若您指定了Skey层级的聚合，则会优先进行Skey层级聚合（效果与EXTS.S.MRANGE命令相同），再从Pkey层级对第一次聚合结果进行二次聚合。

选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。
	• fromTs：查询的开始时间（Unix时间戳），单位为毫秒。
	• toTs：查询的结束时间（Unix时间戳），单位为毫秒，支持用 `*` 表示系统当前的时间戳，若该值等于fromTs可实现单时间点查询。
	• pkeyAggregationType：Pkey的聚合类型，更多信息请参见聚合功能语法。
	• pkeyTimeBucket：Pkey的采样间隔，单位为毫秒，最小值为1,000毫秒。
	Tair会将该时间范围内的数据进行聚合并返回一个结果，返回的时间点为采样间隔的开始时间。
	• MAXCOUNT：指定每个Skey返回的Datapoint条数，默认为不填（Tair的上限为1,000,000条）。
	• AGGREGATION：
	  ◦ aggregationType：Skey的聚合类型，更多信息请参见聚合功能语法。
	  ◦ timeBucket：Skey的采样间隔，单位为毫秒，最小值为1,000毫秒。
	Tair会将该时间范围内的数据进行聚合并返回一个结果，返回的时间点为采样间隔的开始时间。
	• WITHLABELS：设置返回结果中是否包含标签信息，默认为不填（不显示标签信息）。
	• filter：过滤条件，您可以根据Skey的标签（LABELS）过滤目标Skey，更多信息请参见索引过滤语法。
	
	说明 构建filter时，必须存在EQ、CONTAINS、LIST_MATCH逻辑中的任意一个，否则会查询失败。

类别	说明
返回值	• 执行成功：返回聚合结果。 • nil：表示Pkey或Skey不存在。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.P.RANGE foo 1644451031662 * SUM 500000 AGGREGATION SUM 10000 FILTER sensor_id=1`
	返回示例：
	`1) 1) 1) (integer) 1644459500000`
	`2) "40"`
	`2) 1) (integer) 1644460500000`
	`2) "29"`
	`3) 1) (integer) 1644481000000`
	`2) "30"`
	`2) (integer) 0`

EXTS.S.RAW_MODIFY

类别	说明
语法	`EXTS.S.RAW_MODIFY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
时间复杂度	O(1)
命令描述	修改指定Skey中Datapoint数据的value值。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。 • ts：待更新Datapoint的Unix时间戳，单位为毫秒。 • value：待更新的值，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如`LABELS sensor_id 1`。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.RAW_MODIFY foo temperature 1644310456023 31.5`
	返回示例：
	`OK`

EXTS.S.RAW_MMODIFY

类别	说明
语法	`EXTS.S.RAW_MMODIFY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
时间复杂度	O(n)，其中n为keynumber。
命令描述	批量修改多个指定Skey中Datapoint数据的value值。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • keynumber：指定多条数据的个数。 • Skey：Skey名称。 • ts：待更新Datapoint的Unix时间戳，单位为毫秒。 • value：待更新的值，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.RAW_MMODIFY foo 3 temperature 1644565954814 30.2 pressure 1644565954814 2.05 distance 1644565954814 0.5`
	返回示例：
	`1) OK`
	`2) OK`
	`3) OK`

EXTS.S.RAW_INCRBY

类别	说明
语法	`EXTS.S.RAW_INCRBY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
时间复杂度	O(1)
命令描述	该命令中的value值会与指定Skey中Datapoint数据的value相加实现递增，也可以指定该命令中的value为负数实现递减。若Pkey或Skey不存在则会自动创建，默认初始值为0，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • Skey：Skey名称。 • ts：待更新Datapoint的Unix时间戳，单位为毫秒。 • value：待增加操作的值，可以指定该值为负数实现相减，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。

类别	说明
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	提前执行 `EXTS.S.ADD foo temperature 1644310456 30.0` 命令。
	命令示例：
	`EXTS.S.RAW_INCRBY foo temperature 1644310456 3.3`
	返回示例：
	`OK`
	若此时执行 `EXTS.S.GET foo temperature` 命令，将会返回如下结果：
	`1) (integer) 1644310456`
	`2) "33.3"`

EXTS.S.RAW_MINCRBY

类别	说明
语法	`EXTS.S.RAW_MINCRBY` *`Pkey keynumber Skey ts value`* `[`*`Skey ts value ...`*`] [DATA_ET` *`time`*`] [CHUNK_SIZE` *`size`*`] [UNCOMPRESSED] [LABELS` *`label1 val1 ...`*`]`
时间复杂度	O(n)，其中n为keynumber。
命令描述	批量修改多个指定Skey中Datapoint数据的value值，该命令中的value值会与指定Skey中Datapoint数据的value相加实现递增，也可以指定该命令中的value为负数实现递减。若Pkey或Skey不存在则会自动创建，属性（过期时间、是否开启压缩等）仅在Skey不存在并自动创建的情况下生效。
选项	• Pkey：PKey名称（TairTS数据结构），用于指定命令调用的TairTS对象。 • keynumber：指定多条数据的个数。 • Skey：Skey名称。 • ts：待更新Datapoint的Unix时间戳，单位为毫秒。 • value：待增加操作的值，可以指定该值为负数实现相减，数据类型为双精度浮点（double）型。 • DATA_ET time：DataPoint数据的相对过期时间，单位为毫秒，默认为不填（表示不会过期）。 • CHUNK_SIZE：单个Chunk可保存的Datapoint数，默认值为256条，取值范围为[1,256]。 • UNCOMPRESSED：设置Skey不开启压缩，默认为不填（即开启压缩）。 • LABELS：Skey的属性，输入一组或多组对应的标签名、标签值，例如 `LABELS sensor_id 1` 。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`EXTS.S.RAW_MMODIFY foo 3 temperature 1644565954814 30.2 pressure 1644565954814 2.05 distance 1644565954814 0.5`
	返回示例：
	`1) OK`
	`2) OK`
	`3) OK`

索引过滤语法

您可以根据Skey的标签（LABELS）过滤目标Skey。过滤条件（filter）的语法如下：

> **说明** 构造filter时，支持如下所有命令及组合使用，但必须存在EQ、CONTAINS、LIST_MATCH逻辑中的任意一个。

filter命令	说明	逻辑
`L = V`	标签L等于V。	EQ（equals）
`L !=`	标签L不为NULL，即目标Skey包含标签L。	CONTAINS
`L = (v1,v2,...)`	标签L为v1或v2等。	LIST_TMATCH
`L != V`	标签L不等于V。	NOEQ（equals）
`L =`	标签L为NULL，即目标Skey不包含标签L。	NOCONTAINS
`L != (v1,v2,...)`	标签L不为v1和v2等。	LIST_NOTMATCH

聚合功能语法

聚合操作会对采集间隔（timeBucket）范围内的数据进行聚合，支持如下聚合类型：

- `MAX`：最大值
- `MIN`：最小值
- `AVG`：平均值
- `SUM`：求和
- `FIRST`：第一个值
- `LAST`：最后一个值
- `RANGE`：范围（最大值 - 最小值）
- `COUNT`：value数量
- `STD.P`：总体方差
- `STD.S`：样本方差
- `VAR.P`：总体标准差
- `VAR.S`：样本标准差

3.6.9. TairCpc

TairCpc是基于CPC（Compressed Probability Counting）压缩算法开发的数据结构，支持仅占用很小的内存空间对采样数据进行高性能计算。

背景信息

在大数据实时决策场景中，通常会将业务日志流入实时计算系统完成计算，然后将计算结果存储至在线存储系统，最终由实时规则或决策系统进行决策，例如：

- 防控信用卡欺诈交易：快速判断刷卡环境是否可信，若发现异常则需第一时间拦截交易。
- 防控黄牛团伙恶意牟利：实时识别并阻止通过虚拟设备、虚假地址等方式损害平台利益的行为。

您可以利用TairCpc将实时数据，按不同的去重维度，结构化地存储到Tair数据库中，即可在高速的访问场景中直接获得结果，实现存储、计算一体化。同时TairCpc提供多重聚合运算，可以在纳秒级聚合数据结果，具备实时风控的能力。

TairCpc简介

CPC是一种高性能数据去重算法，可以将不同的值作为数据流进行计数，支持将多个数据块合并、去重，获得去重后的总计数。相比HLL（Hyperloglog）算法，在相同精度下，CPC大约可节省40%内存空间。

同时，TairCpc在开源CPC算法的基础上，将误差率优化至0.008%（开源CPC为0.67%；HLL误差率为1.95%）。

主要特征

- 内存占用低，支持增量读写，实现IO最小化。

- 高性能去重，同时拥有超高去重精度。
- 误差率稳定收敛。

典型场景

- 银行安全系统
- 秒杀限购
- 防控用户（或黄牛团伙）恶意牟利

前提条件

实例为Tair（Redis企业版）：

- 性能增强型（小版本为1.7.20及以上）
- 持久内存型（小版本为1.2.3.3及以上）

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

操作对象为Tair（Redis企业版）实例中的TairCpc数据。

命令列表

TairCpc命令

命令	语法	说明
CPC.UPDATE	`CPC.UPDATE key item [EX|EXAT|PX|PXAT time]`	在指定TairCpc中添加item。若TairCpc不存在则自动新建，若待添加的item已存在于目标TairCpc中，则不会进行操作。
CPC.ESTIMATE	`CPC.ESTIMATE key`	获取指定TairCpc去重后的基数估算值，返回值的数据类型为double类型，您可以仅取整数部分（忽略小数点后的数据）。
CPC.UPDATE2EST	`CPC.UPDATE2EST key item [EX|EXAT|PX|PXAT time]`	在指定TairCpc中添加item，返回更新后的基数估算值。若TairCpc不存在则自动新建。
CPC.UPDATE2JUD	`CPC.UPDATE2JUD key item [EX|EXAT|PX|PXAT time]`	在指定TairCpc中添加item，并返回更新后的基数估算值和其与更新前的差值。若返回的差值为1则表示写入成功且不存在重复；若为0则表示已存在当前item。若TairCpc不存在则自动新建。
CPC.ARRAY.UPDATE	`CPC.ARRAY.UPDATE key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`	在指定TairCpc中，向目标timestamp对应的时间窗口添加item。若TairCpc不存在则自动新建，`SIZE` 为时间窗口个数，`WIN` 为时间窗口的长度（单位为毫秒）。随着流式数据的写入，TairCpc会持续向前更新并保存 `SIZE` * `WIN` 时间范围内的数据，超过该时间范围的数据会被覆盖、删除。`SIZE` 和 `WIN` 属性仅在新建TairCpc的时生效。 **说明** 例如目标key为计算近10分钟内每分钟的数据：可以设置 `SIZE` 为10（10个时间窗口）、`WIN` 为60000（每个时间窗口为1分钟）。当目标key中写入第11分钟的数据时，第1分钟的数据会逐渐被覆盖、删除。
CPC.ARRAY.ESTIMATE	`CPC.ARRAY.ESTIMATE key timestamp`	获取指定TairCpc中目标timestamp所在时间窗口的基数估算值。
CPC.ARRAY.ESTIMATE.RANGE	`CPC.ARRAY.ESTIMATE.RANGE key start_time end_time`	获取指定TairCpc的指定时间段内（包含指定时间点）各个时间窗口的基数估算值。

命令	语法	说明
CPC.ARRAY.ESTIMATE.RANGE.MERGE	`CPC.ARRAY.ESTIMATE.RANGE.MERGE key timestamp range`	获取指定TairCpc在指定时间点及往后的range时间段内，时间窗口合并、去重后的基数估算值。
CPC.ARRAY.UPDATE2EST	`CPC.ARRAY.UPDATE2EST key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`	在指定TairCpc中，向目标timestamp对应的时间窗口添加item，并返回该时间窗口更新后的基数估算值。若TairCpc不存在则自动新建，新建参数用法与CPC.ARRAY.UPDATE一致。
CPC.ARRAY.UPDATE2JUD	`CPC.ARRAY.UPDATE2JUD key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`	在指定TairCpc中，向目标timestamp对应的时间窗口添加item，并返回该时间窗口更新后的基数估算值和其与更新前的差值。若返回的差值为1，则表示写入成功且不存在重复；若为0则表示已存在当前item。若TairCpc不存在则自动新建，新建参数用法与CPC.ARRAY.UPDATE一致。
DEL	`DEL key [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairCpc数据。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

CPC.UPDATE

类别	说明
语法	`CPC.UPDATE key item [EX|EXAT|PX|PXAT time]`
时间复杂度	O(1)
命令描述	在指定TairCpc中添加item。若TairCpc不存在则自动新建，若待添加的item已存在于目标TairCpc中，则不会进行操作。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • item：待添加的数据。 • EX：指定key的相对过期时间，单位为秒，不传此参数表示不过期。 • EXAT：指定key的绝对过期时间（Unix时间戳），单位为秒，不传此参数表示不过期。 • PX：指定key的相对过期时间，单位为毫秒，不传此参数表示不过期。 • PXAT：指定key的绝对过期时间（Unix时间戳），单位为毫秒，不传此参数表示不过期。
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.UPDATE foo f1 EX 3600`
	返回示例：
	`OK`

CPC.ESTIMATE

类别	说明
语法	`CPC.ESTIMATE key`
时间复杂度	O(1)
命令描述	获取指定TairCpc去重后的基数估算值，返回值的数据类型为double类型，您可以仅取整数部分（忽略小数点后的数据）。
选项	• key：Key名称（TairCpc数据结构）。
返回值	• 执行成功：返回估算值，数据类型为double类型。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.ESTIMATE foo`
	返回示例：
	`"19.000027716212127"`

CPC.UPDATE2EST

类别	说明
语法	`CPC.UPDATE2EST key item [EX|EXAT|PX|PXAT time]`
时间复杂度	O(1)
命令描述	在指定TairCpc中添加item，返回更新后的基数估算值。若TairCpc不存在则自动新建。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • item：待添加的数据。 • EX：指定key的相对过期时间，单位为秒，不传此参数表示不过期。 • EXAT：指定key的绝对过期时间（Unix时间戳），单位为秒，不传此参数表示不过期。 • PX：指定key的相对过期时间，单位为毫秒，不传此参数表示不过期。 • PXAT：指定key的绝对过期时间（Unix时间戳），单位为毫秒，不传此参数表示不过期。
返回值	• 执行成功：返回更新后的估算值，数据类型为double类型。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.UPDATE2EST foo f3`
	返回示例：
	`"3.0000004768373003"`

CPC.UPDATE2JUD

类别	说明
语法	`CPC.UPDATE2JUD key item [EX|EXAT|PX|PXAT time]`
时间复杂度	O(1)

类别	说明
命令描述	在指定TairCpc中添加item，并返回更新后的基数估算值和其与更新前的差值。若返回的差值为1则表示写入成功且不存在重复；若为0则表示已存在当前item。若TairCpc不存在则自动新建。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • item：待添加的数据。 • EX：指定key的相对过期时间，单位为秒，不传此参数表示不过期。 • EXAT：指定key的绝对过期时间（Unix时间戳），单位为秒，不传此参数表示不过期。 • PX：指定key的相对过期时间，单位为毫秒，不传此参数表示不过期。 • PXAT：指定key的绝对过期时间（Unix时间戳），单位为毫秒，不传此参数表示不过期。
返回值	• 执行成功：返回更新后的估算值和该值与更新前的差值，数据类型均为double类型。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.UPDATE2JUD foo f20`
	返回示例：
	`1) "20.000027716212127"  // 更新后，TairCpc的估算值为20。`
	`2) "1.0000014901183398"  // 20 - 19 = 1`

CPC.ARRAY.UPDATE

类别	说明
语法	`CPC.ARRAY.UPDATE key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`
时间复杂度	O(1)
命令描述	在指定TairCpc中，向目标timestamp对应的时间窗口添加item。若TairCpc不存在则自动新建，`SIZE`为时间窗口个数，`WIN`为时间窗口的长度（单位为毫秒）。随着流式数据的写入，TairCpc会持续向前更新并保存`SIZE`*`WIN`时间范围内的数据，超过该时间范围的数据会被覆盖、删除。`SIZE`和`WIN`属性仅在新建TairCpc的时生效。 **说明** 例如目标key为计算近10分钟内每分钟的数据：可以设置`SIZE`为10（10个时间窗口）、`WIN`为60000（每个时间窗口为1分钟）。当目标key中写入第11分钟的数据时，第1分钟的数据会逐渐被覆盖、删除。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • timestamp：指定的Unix时间戳，单位为毫秒。 • item：待添加的数据。 • EX：指定key的相对过期时间，单位为秒，不传此参数表示不过期。 • EXAT：指定key的绝对过期时间（Unix时间戳），单位为秒，不传此参数表示不过期。 • PX：指定key的相对过期时间，单位为毫秒，不传此参数表示不过期。 • PXAT：指定key的绝对过期时间（Unix时间戳），单位为毫秒，不传此参数表示不过期。 • SIZE：时间窗口个数，默认为10，范围为[1,1000]，建议设置在120以内。 • WIN：时间窗口的长度（单位为毫秒），默认为60000毫秒（1分钟）。

类别	说明
返回值	• OK：表示执行成功。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.ARRAY.UPDATE foo 1645584510000 f1 SIZE 120 WIN 10000`
	返回示例：
	`OK`

CPC.ARRAY.ESTIMATE

类别	说明
语法	`CPC.ARRAY.ESTIMATE key timestamp`
时间复杂度	O(1)
命令描述	获取指定TairCpc中目标timestamp所在时间窗口的基数估算值。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • timestamp：指定的Unix时间戳，单位为毫秒。
返回值	• 执行成功：返回对应时间窗口的基数估算值。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.ARRAY.ESTIMATE foo 1645584532000`
	返回示例：
	`"2"`

CPC.ARRAY.ESTIMATE.RANGE

类别	说明
语法	`CPC.ARRAY.ESTIMATE.RANGE key start_time end_time`
时间复杂度	O(1)
命令描述	获取指定TairCpc的指定时间段内（包含指定时间点）各个时间窗口的基数估算值。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • start_time：查询的开始时间（Unix时间戳），单位为毫秒。 • end_time：查询的结束时间（Unix时间戳），单位为毫秒。
返回值	• 执行成功：返回目标时间窗口的基数估算值。 • 其它情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`CPC.ARRAY.ESTIMATE.RANGE foo 1645584510000 1645584550000`
	返回示例：
	`1) "2"`
	`2) "0"`
	`3) "1"`
	`4) "0"`
	`5) "0"`

CPC.ARRAY.ESTIMATE.RANGE.MERGE

类别	说明
语法	`CPC.ARRAY.ESTIMATE.RANGE.MERGE key timestamp range`
时间复杂度	O(1)
命令描述	获取指定TairCpc在指定时间点及往后的range时间段内，时间窗口合并、去重后的基数估算值。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • timestamp：查询的开始时间（Unix时间戳），单位为毫秒。 • range：查询的时间范围（Unix时间戳），单位为毫秒。
返回值	• 执行成功：返回目标key在指定时间段内去重后的基数估算值。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.ARRAY.ESTIMATE.RANGE.MERGE foo 1645584510000 100000`
	返回示例：
	`"6"`

CPC.ARRAY.UPDATE2EST

类别	说明
语法	`CPC.ARRAY.UPDATE2EST key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`
时间复杂度	O(1)
命令描述	在指定TairCpc中，向目标timestamp对应的时间窗口添加item，并返回该时间窗口更新后的基数估算值。若TairCpc不存在则自动新建，新建参数用法与CPC.ARRAY.UPDATE一致。

类别	说明
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • timestamp：指定的Unix时间戳，单位为毫秒。 • item：待添加的数据。 • EX：指定key的相对过期时间，单位为秒，不传此参数表示不过期。 • EXAT：指定key的绝对过期时间（Unix时间戳），单位为秒，不传此参数表示不过期。 • PX：指定key的相对过期时间，单位为毫秒，不传此参数表示不过期。 • PXAT：指定key的绝对过期时间（Unix时间戳），单位为毫秒，不传此参数表示不过期。 • SIZE：时间窗口个数，默认为10，范围为[1,1000]，建议设置在120以内。 • WIN：时间窗口的长度（单位为毫秒），默认为60000毫秒（1分钟）。
返回值	• 执行成功：返回目标时间窗口更新后的估算值。 • 其它情况返回相应的异常信息。

示例	命令示例：
	`CPC.ARRAY.UPDATE2EST foo 1645584530000 f3`
	返回示例：
	`"3"`

CPC.ARRAY.UPDATE2JUD

类别	说明
语法	`CPC.ARRAY.UPDATE2JUD key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`
时间复杂度	O(1)
命令描述	在指定TairCpc中，向目标timestamp对应的时间窗口添加item，并返回该时间窗口更新后的基数估算值和其与更新前的差值。若返回的差值为1，则表示写入成功且不存在重复；若为0则表示已存在当前item。若TairCpc不存在则自动新建，新建参数用法与CPC.ARRAY.UPDATE一致。
选项	• key：Key名称（TairCpc数据结构），用于指定命令调用的TairCpc对象。 • item：待添加的数据。 • EX：指定key的相对过期时间，单位为秒，不传此参数表示不过期。 • EXAT：指定key的绝对过期时间（Unix时间戳），单位为秒，不传此参数表示不过期。 • PX：指定key的相对过期时间，单位为毫秒，不传此参数表示不过期。 • PXAT：指定key的绝对过期时间（Unix时间戳），单位为毫秒，不传此参数表示不过期。 • SIZE：时间窗口个数，默认为10，范围为[1,1000]，建议设置在120以内。 • WIN：时间窗口的长度（单位为毫秒），默认为60000毫秒（1分钟）。
返回值	• 执行成功：返回目标时间窗口更新后的估算值和该值与更新前的差值。 • 其它情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`CPC.ARRAY.UPDATE2JUD foo 1645584530000 f7`
	返回示例：
	`1) "8"  // 更新后，TairCpc的估算值为8。`
	`2) "1"  // 8 - 7 = 1`

3.6.10. TairZset

通过TairZset可实现256维度的double类型的分值排序，适用于游戏、直播、音乐、电商等行业的排行榜场景，可极大提升数据处理效率，且客户端适配简易，无需任何编解码封装。

TairZset简介

原生Redis支持的排序结构Sorted Set（也称Zset）只支持1个double类型的分值排序，实现多维度排序时较为困难。例如通过IEEE 754结合拼接的方式实现多维度排序，此类方式存在实现复杂、精度下降、**EXZINCRBY**命令无法使用等局限性。

借助阿里云自研的TairZset数据结构，可帮助您轻松实现多维度排序能力，相较于传统方案具有如下优势：

- 最大支持256维的double类型的分值排序（排序优先级为从左往右）。
  对于多维score而言，左边的score优先级大于右边的score，以一个三维score为例：score1#score2#score3，TairZset在比较时，会先比较score1，只有score1相等时才会比较score2，否则就以score1的比较结果作为整个score的比较结果。同样，只有当score2相等时才会比较score3。若所有维度分数都相同，则会按照元素顺序（ascii顺序）进行排序。
  为了方便理解，可以把#想象成小数点（.），例如0#99、99#90和99#99大小关系可以理解为0.99 < 99.90 < 99.99，即0#99 < 99#90 < 99#99。
- 支持**EXZINCRBY**命令，不再需要取回当前数据，在本地增加值后再拼接写回Redis。
- 支持和原生Zset相似的API。
- 提供和的能力。普通排行榜分布式架构排行榜
- 提供开源TairJedis客户端，无需任何编解码封装，您也可以参考开源自行实现封装其他语言版本。

典型场景

适用于游戏、直播、音乐、电商等行业的排行榜场景，例如：

- 直播排行榜：直播PK中，主播之间先按照当前人气值排序；如果人气值相同，再按照点赞数排序；如果点赞数也相同，再按照礼物金额进行排序等。
- 奖牌排行榜：从金、银、铜牌的维度对参赛方进行排名，先按照金牌数量排序；如果金牌数量一致，再以银牌数量排序；如果银牌数量也一致，再按照铜牌数量排序。
- 游戏排行榜：玩家之间按照得分、任务完成时长、段位等多个维度进行排名。

该Module已开源，更多信息请参见TairZset。

最佳实践

- 基于TairZset轻松实现多维排行榜
- 基于TairZset实现分布式架构排行榜

前提条件

实例为Tair（Redis企业版）性能增强型，且小版本为1.7.1及以上。

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

操作对象为性能增强型实例中的TairZset数据。

命令列表

命令	语法	说明
EXZADD	`EXZADD key [NX|XX] [CH] [INCR] score member [score member ...]`	将指定的分数与成员信息存储到TairZset结构的Key中，支持指定多个分数与成员。 **说明** 如需实现多维度的排序，各维度的分数之间使用井号（#）分隔，例如 `111#222#121` ，且要求该Key中所有成员的分数格式必须相同。
EXZINCRBY	`EXZINCRBY key increment member`	为Key（TairZset数据结构）中的成员增加分数，increment为要增加的分数值。
EXZSCORE	`EXZSCORE key member`	返回存储在Key（TairZset数据结构）中成员的分数，如果Key或Key中的成员不存在，系统会返回nil。
EXZRANGE	`EXZRANGE key min max [WITHSCORES]`	返回存储在Key（TairZset数据结构）中指定范围的元素。
EXZREVRANGE	`EXZREVRANGE key min max [WITHSCORES]`	返回存储在Key（TairZset数据结构）中指定范围内的元素，元素按分值从高到低的顺序排列，按字典序降序排列分数相同的元素。 **说明** 除排序方式相反外，本命令和EXZRANGE用法相似。
EXZRANGEBYSCORE	`EXZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]`	返回存储在Key（TairZset数据结构）中，分数大于等于min且小于等于max值的所有元素，返回元素按分数从低到高排列，分数相同的元素按照字典顺序返回。
EXZREVRANGEBYSCORE	`EXZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]`	返回存储在Key（TairZset数据结构）中，分数大于等于min且小于等于max值的所有元素。和TairZset中元素默认排序相反，该命令返回元素按照分数从高到低排列，分数相同的元素按照逆字典序排列。 **说明** 除排序方式相反外，本命令和EXZRANGEBYSCORE用法相似，注意本命令中max在前。
EXZRANGEBYLEX	`EXZREMRANGEBYLEX key min max`	为确保元素按照字典序排列，当Key中所有元素分数相同时，则该命令返回存储在Key中介于min和max值之间的元素。
EXZREVRANGEBYLEX	`EXZREVRANGEBYLEX key max min [LIMIT offset count]`	为确保元素按照字典序排列，当Key中所有元素分数相同时，则该命令返回存储在Key中介于max和min之间的元素。 **说明** 除排序方式相反外，本命令和EXZRANGEBYLEX用法相同，注意本命令中max在前。
EXZREM	`EXZREM key member [member ...]`	移除存储Key中的指定成员，如果指定成员不存在，则忽略。
EXZREMRANGEBYSCORE	`EXZREMRANGEBYSCORE key min max`	移除存储在Key（TairZset数据结构）中，分数大于等于min且小于等于max值的元素。
EXZREMRANGEBYRANK	`EXZREMRANGEBYRANK key start stop`	移除存储在Key（TairZset数据结构）中，级别介于start和stop之间的元素。

命令	语法	说明
EXZREMRANGEBYLEX	`EXZREMRANGEBYLEX key min max`	为确保元素按照字典序排列，当Key中所有元素分数相同时，则该命令移除存储在Key中介于max和min之间的元素。 **说明** 若使用相同的min和max参数值执行该命令和EXZRANGEBYLEX命令，则该命令移除的元素与EXZRANGEBYLEX命令返回的元素相同。
EXZCARD	`EXZCARD key`	返回存储在Key（TairZset数据结构）中的基数（即元素的个数）。
EXZRANK	`EXZRANK key member`	返回存储在Key（TairZset数据结构）中的成员的级别，按照分数由低到高排列。级别（或索引）从0开始计数，即分数最低的成员的级别为0。
EXZREVRANK	`EXZREVRANK key member`	返回存储在Key（TairZset数据结构）中的成员的级别。返回结果按照分数从高到低排列。 级别（或索引）从0开始计数，即分数最高的成员的级别为0。 **说明** 除排序规则相反外，本命令和EXZRANK用法类似。
EXZCOUNT	`EXZCOUNT key min max`	返回存储在Key（TairZset数据结构）中，分数介于min和max之间的元素的个数。
EXZLEXCOUNT	`EXZLEXCOUNT key min max`	为确保元素按照字典序排列，若Key中所有元素分数相同，则该命令返回存储在Key，值介于min和max之间的元素的数量。
EXZRANKBYSCORE	`EXZRANKBYSCORE key score`	计算指定的分数在Key（TairZset数据结构）中按照分数从低到高排序的排名位置。级别（或索引）从0开始计数，即分数最低的成员的级别为0。 **说明** 若指定的分数不存在，则返回该分数在Key（TairZset数据结构）中的预计排名；若指定的分数已存在，Tair会默认将指定的分数排在已存在的分数之前。
EXZREVRANKBYSCORE	`EXZREVRANKBYSCORE key score`	计算指定的分数在Key（TairZset数据结构）中按照分数从高到低排序的排名位置。级别（或索引）从0开始计数，即分数最高的成员的级别为0。 **说明** 若指定的分数不存在，则返回该分数在Key（TairZset数据结构）中的预计排名；若指定的分数已存在，Tair会默认将指定的分数排在已存在的分数之后。
DEL	`DEL key [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairZset数据。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

EXZADD

类别	说明
语法	`EXZADD key [NX|XX] [CH] [INCR] score member [score member ...]`

类别	说明
时间复杂度	O(N)
命令描述	将指定的分数与成员信息存储到TairZset结构的Key中，支持指定多个分数与成员，系统会根据Key和成员是否存在，执行不同的策略： **说明** 如需实现多维度的排序，各维度的分数之间使用井号（#）分隔，例如 `111#222#121` ，且要求该Key中所有成员的分数格式必须相同。 • 如果指定的Key存在，但其数据结构不是TairZset，系统将返回错误。 • 如果指定的Key不存在，系统将创建一个TairZset结构的Key，然后将指定的成员添加至该Key中。 • 如果指定的成员已经是TairZset的成员，则会更新该成员的分数，并将该成员重新插入到正确的位置，避免打乱排序。 • 每个分数值使用双精度浮点数的字符串表示，+inf和-inf值都是有效值。
选项	• NX：只添加新元素，不更新已经存在的元素。 • XX：只更新已经存在的元素，不添加新的元素。 • CH：一般情况下，本命令的返回值为添加的新元素数量，通过该参数可以将返回值改为发生变化的元素总数。 **说明** 发生变化的元素包含新元素和分数有更新的已有元素。 因此，如果命令行中已存在的一个元素的分数没有发生变化，则该元素不算作发生变化的元素。 • INCR：指定此选项时，EXZADD的行为与EXZINCRBY类似，即该模式下仅支持指定一对分数与元素。
返回值	返回值为整数数字，具体为： • 未指定任何选项时，返回值为添加到Key中的元素数量（不包括仅更新分数的元素）。 • 指定了CH选项时，返回值为发生变化的（新增或更新）元素数量。 • 指定了INCR选项时，则返回值为成员的新分数（字符串形式）。如果使用了多维度分数，则该成员分数的格式为 `格式为“分数1#分数2#分数3#...”` ，例如 `2#0#6` 。 **说明** 如果停止该操作（命令中包含XX或NX选项），则返回nil。

示例	命令示例：
	`EXZADD testkey NX 1#0#3 a 1#0#2 b`
	返回示例：
	`(integer) 2`

EXZINCRBY

类别	说明
语法	`EXZINCRBY key increment member`
时间复杂度	O(log(N))

类别	说明
命令描述	为Key（TairZset数据结构）中的成员增加分数，increment为要增加的分数值，系统会根据Key和成员是否存在，执行不同的策略： • 如果指定的Key存在，但其数据结构不是TairZset，系统将返回错误。 • 如果指定的Key不存在，系统将创建一个TairZset结构的Key，然后将指定的成员作为该Key的唯一成员。 • 如果指定的Key没有成员，系统将向其中添加一个分数为increment参数值的成员，即假设成员原分数为0.0分。 **说明** • 如需实现多维度的排序，各维度的分数之间使用井号（#）分隔，例如 `111#222#121` ，且要求该Key中所有成员的分数格式必须相同。 • 分数值应为数字值的字符串形式，可以为双精度浮点数。 如果需要降低成员的分数，则指定一个负数。
选项	无
返回值	返回成员的新分数（字符串形式），如果使用了多维度分数，则该成员分数的格式为 `格式为"分数1#分数2#分数3#..."` ，例如 `2#0#6` 。

示例	命令示例：
	`EXZINCRBY testkey 2#2#1 a`
	返回示例：
	`"3#2#4"`

EXZSCORE

类别	说明
语法	`EXZSCORE key member`
时间复杂度	O(1)
命令描述	返回存储在Key（TairZset数据结构）中成员的分数，如果Key或Key中的成员不存在，系统会返回nil。
选项	无
返回值	返回成员的分数（字符串形式），如果使用了多维度分数，则该成员分数的格式为 `格式为"分数1#分数2#分数3#..."` ，例如 `2#0#6` 。

示例	命令示例：
	`EXZSCORE testkey a`
	返回示例：
	`"3#2#4"`

EXZRANGE

类别	说明
语法	`EXZRANGE key min max [WITHSCORES]`

类别	说明
时间复杂度	O(log(N)+M)，其中，N表示TairZset中元素的数量，M表示返回的元素的数量。
命令描述	返回存储在Key（TairZset数据结构）中指定范围的元素。

选项	• min、max：代表基于0的索引值。其中，第一个元素的索引值为0，第二个元素的索引值为1，依此类推。 可使用这两个参数指定一个闭区间。
	说明
	◦ 如果索引值为负数，则表示返回的元素末尾往前偏移的量。比如，-1代表Key的最后一个元素，-2代表倒数第二个元素，依此类推。
	◦ 如需查询所有的元素信息，min取值为0，max取值为-1。
	◦ 如果min的值比Key中最后一个元素的索引值或max的值更大，则返回空列表。
	• WITHSCORES：返回值中包含元素的分数，即返回列表的数据格式为 `值1，分数1，...，值N，分数N` ，例如：
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

返回值	返回指定范围内元素的列表，如果使用了WITHSCORES选项，则返回结果中包含元素的分数。

示例	命令示例：
	`EXZRANGE testkey 0 -1 WITHSCORES`
	返回示例：
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

EXZREVRANGE

类别	说明
语法	`EXZREVRANGE key min max [WITHSCORES]`
时间复杂度	O(log(N)+M)，其中，N表示TairZset中元素的数量，M表示返回的元素的数量。
命令描述	返回存储在Key（TairZset数据结构）中指定范围内的元素，元素按分值从高到低的顺序排列，按字典序降序排列分数相同的元素。 **说明** 除排序方式相反外，本命令和EXZRANGE用法相似。

类别	说明

选项	• min、max：代表基于0的索引值。其中，第一个元素的索引值为0，第二个元素的索引值为1，依此类推。 可使用这两个参数指定一个闭区间。
	说明
	◦ 如果索引值为负数，则表示返回的元素末尾往前偏移的量。比如，-1代表Key的最后一个元素，-2代表倒数第二个元素，依此类推。
	◦ 如需查询所有的元素信息，min取值为0，max取值为-1。
	◦ 如果min的值比Key中最后一个元素的索引值或max的值更大，则返回空列表。
	• WITHSCORES：返回值中包含元素的分数，即返回列表的数据格式为 `值1，分数1，...，值N，分数N` ，例如：
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

返回值	返回指定范围内的元素列表，如果使用了WITHSCORES选项，则返回结果中包含元素的分数。

示例	命令示例：
	`EXZREVRANGE testkey 0 -1 WITHSCORES`
	返回示例：
	`1) "a"`
	`2) "3#2#4"`
	`3) "b"`
	`4) "1#0#2"`

EXZRANGEBYSCORE

类别	说明
语法	`EXZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]`
时间复杂度	O(log(N)+M)，其中，N表示TairZset中元素的数量，M表示返回的元素的数量。 **说明** 当M为常量时（例如使用LIMIT选项指定总是返回前10个元素时），可将该公式看作O(log(N))。
命令描述	返回存储在Key（TairZset数据结构）中，分数大于等于min且小于等于max值的所有元素，返回元素按分数从低到高排列，分数相同的元素按照字典顺序返回。

类别	说明
选项	• min、max：分别表示最小分数和最大分数，如Key中的元素采用了多维度的分数，各维度的分数之间使用井号（#）分隔。 **说明** ◦ 在不确定Key中元素的最高分和最低分的情况下，如果想要查询Key中分数大于等于或小于等于某一特定值的元素，请将min和max分别设置为负无穷大（ `-inf` ）和正无穷大（ `+inf` ）。 ◦ 默认数据范围为闭区间，如需指定开区间，则在分数范围前添加半角圆括号，例如 `(1 5` 表示返回分数大于1且小于等于5的元素。 • WITHSCORES：返回值中包含元素的分数。 • LIMIT offset count：指定返回结果的数量及区间，如果count为负数，则返回从offset开始的所有元素。 **说明** 如果offset较大，则需要遍历整个Key以定位到offset元素，然后才能返回元素，即会增加时间复杂度。
返回值	返回指定分数范围内的元素列表，如果使用了WITHSCORES选项，则返回结果中包含元素的分数。

命令示例	命令示例：
	`EXZRANGEBYSCORE testkey 0#0#0 6#6#6 WITHSCORES`
	返回示例：
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

EXZREVRANGEBYSCORE

类别	说明
语法	`EXZREVRANGEBYSCORE` *key max min* `[WITHSCORES] [LIMIT` *offset count*`]`
时间复杂度	O(log(N)+M)，其中，N表示TairZset中元素的数量，M表示返回的元素的数量。 **说明** 当M为常量时（比如使用LIMIT选项指定总是返回前10个元素时），可将该公式看作O(log(N))。
命令描述	返回存储在Key（TairZset数据结构）中，分数大于等于min且小于等于max值的所有元素。和TairZset中元素默认排序相反，该命令返回元素按照分数从高到低排列，分数相同的元素按照逆字典序排列。 **说明** 除排序方式相反以外，本命令和EXZRANGEBYSCORE用法相似，注意本命令中max在前。

类别	说明
选项	• min、max：分别表示最小分数和最大分数，如Key中的元素采用了多维度的分数，各维度的分数之间使用井号（#）分隔。 **说明** ◦ 在不确定Key中元素的最高分和最低分的情况下，如果想要查询Key中分数大于等于或小于等于某一特定值的元素，请将min和max分别设置为负无穷大（ `-inf` ）和正无穷大（ `+inf` ）。 ◦ 默认数据范围为闭区间，如需指定开区间，则在分数范围前添加半角圆括号，例如 `(1 5` 表示返回分数大于1且小于等于5的元素。 • WITHSCORES：返回值中包含元素的分数。 • LIMIT offset count：指定返回结果的数量及区间，如果count为负数，则返回从offset开始的所有元素。 **说明** 如果offset较大，则需要遍历整个Key以定位到offset元素，然后才能返回元素，即会增加时间复杂度。
返回值	返回指定分数范围内的元素列表，如果使用了WITHSCORES选项，则返回结果中包含元素的分数。

命令示例	命令示例：
	`EXZREVRANGEBYSCORE testkey 6#6#6 0#0#0`
	返回示例：
	`1) "a"`
	`2) "3#2#4"`
	`3) "b"`
	`4) "1#0#2"`

EXZRANGEBYLEX

类别	说明
语法	`EXZRANGEBYLEX key min max [LIMIT offset count]`
时间复杂度	O(log(N)+M)，其中，N表示TairZset中元素的数量，M表示返回的元素的数量。 **说明** 当M为常量时（例如使用LIMIT选项指定返回前10个元素时），可将该公式看作O(log(N))。
命令描述	为确保元素按照字典序排列，当Key中所有元素分数相同时，则该命令返回存储在Key中介于min和max值之间的元素。 **说明** • 如果Key中元素分数不同，则返回元素未知。 • 采用 `memcmp() C` 函数逐个比对两个元素字符串中的字节。根据比对结果，由低到高排列元素。 若两个字符串包含相同的子字符串，那么字符串越长，其分值越高。

类别	说明

选项	• min、max：分别表示成员名称的最小值和最大值（字符串形式），需指定字符的区间，例如
	  ◦ 开区间：在值的前面增加半角圆括号，例如 `(a` 。
	  ◦ 闭区间：在值的前面增加方括号，例如 `[a`
	  说明 正负无穷大分别为 `+` 和 `-` 。
	• LIMIT offset count：指定返回结果的数量及区间，如果count为负数，则返回从offset开始的所有元素。
	  说明 如果offset较大，则需要遍历整个Key以定位到offset元素，然后才能返回元素，即会增加时间复杂度。

返回值	返回元素名称在指定范围内的元素列表。

命令示例	命令示例：
	`EXZRANGEBYLEX zzz [a [b`
	返回示例：
	`1) "aba"`
	`2) "abc"`

EXZREVRANGEBYLEX

类别	说明
语法	`EXZREVRANGEBYLEX key max min [LIMIT offset count]`
时间复杂度	O(log(N)+M)，其中，N表示TairZset中元素的数量，M表示返回的元素的数量。 说明 当M为常量时（例如使用LIMIT选项指定总是返回前10个元素时），可将该公式看作O(log(N))。
命令描述	为确保元素按照字典序排列，当Key中所有元素分数相同时，则该命令返回存储在Key中介于max和min之间的元素。 说明 除排序方式相反外，本命令和EXZRANGEBYLEX用法相同，注意本命令中max在前。

选项	• min、max：分别表示成员名称的最小值和最大值（字符串形式），需指定字符的区间，例如
	  ◦ 开区间：在值的前面增加半角圆括号，例如 `(a` 。
	  ◦ 闭区间：在值的前面增加方括号，例如 `[a`
	  说明 正负无穷大分别为 `+` 和 `-` 。
	• LIMIT offset count：指定返回结果的数量及区间，如果count为负数，则返回从offset开始的所有元素。
	  说明 如果offset较大，则需要遍历整个Key以定位到offset元素，然后才能返回元素，即会增加时间复杂度。

类别	说明
返回值	返回元素名称在指定范围内的元素列表。

命令示例	命令示例：
	`EXZREVRANGEBYLEX zzz [b [a`
	返回示例：
	`1) "abc"`
	`2) "aba"`

EXZREM

类别	说明
语法	`EXZREM key member [member ...]`
时间复杂度	O(M*log(N))，其中，N为TairZset中元素的数量，M为待移除的元素的数量。
命令描述	移除存储Key中的指定成员，如果指定成员不存在，则忽略。 **说明** 如果指定的Key存在，但其数据结构不是TairZset，系统将返回错误。
选项	无
返回值	返回Key中被移除的成员数量，不包含不存在的成员。

命令示例	命令示例：
	`EXZREM testkey a`
	返回示例：
	`(integer) 1`

EXZREMRANGEBYSCORE

类别	说明
语法	`EXZREMRANGEBYSCORE key min max`
时间复杂度	O(log(N)+M)，其中，N为TairZset中元素的数量，M为待移除的元素的数量。
命令描述	移除存储在Key（TairZset数据结构）中，分数大于等于min且小于等于max值的元素。
选项	min和max分别表示最小分数和最大分数，如Key中的元素采用了多维度的分数，各维度的分数之间使用井号（#）分隔。 **说明** • 在不确定Key中元素的最高分和最低分的情况下，如果想要移除Key中分数大于等于或小于等于某一特定值的元素，请将min和max分别设置为负无穷大（`-inf`）和正无穷大（`+inf`）。 • 默认数据范围为闭区间，如需指定开区间，则在分数范围前添加半角圆括号，例如`EXZREMRANGEBYSCORE (1 5`表示删除分数大于1且小于等于5的元素。
返回值	返回被移除的元素的数量。

类别	说明

命令示例	命令示例：
	`EXZREMRANGEBYSCORE testkey 3#2#4 6#6#6`
	返回示例：
	`(integer) 1`

EXZREMRANGEBYRANK

类别	说明
语法	`EXZREMRANGEBYRANK key start stop`
时间复杂度	O(log(N)+M)，其中，N为TairZset中元素的数量，M为该操作移除的元素的数量。
命令描述	移除存储在Key（TairZset数据结构）中，级别介于start和stop之间的元素。
选项	start和stop均为基于零的索引值，其中，0代表分数最低的元素。 当索引值为负数，代表从最高分数元素开始的偏移量，例如-1为分数最高的元素，-2为分数第二高的元素，依此类推。
返回值	被移除的元素的数量。

命令示例	命令示例：
	`EXZREMRANGEBYRANK testkey 0 1`
	返回示例：
	`(integer) 1`

EXZREMRANGEBYLEX

类别	说明
语法	`EXZREMRANGEBYLEX key min max`
时间复杂度	O(log(N)+M)，其中，N为TairZset中元素的数量，M为该操作移除的元素的数量。
命令描述	为确保元素按照字典序排列，当Key中所有元素分数相同时，则该命令移除存储在Key中介于max和min之间的元素。 **说明** 若使用相同的min和max参数值执行该命令和EXZRANGEBYLEX命令，则该命令移除的元素与EXZRANGEBYLEX命令返回的元素相同。
选项	min、max：分别表示成员名称的最小值和最大值（字符串形式），需指定字符的区间，例如 • 开区间：在值的前面增加半角圆括号，例如 `(a` 。 • 闭区间：在值的前面增加方括号，例如 `[a`
返回值	被移除的元素的数量。

类别	说明

命令示例	命令示例：
	`EXZREMRANGEBYLEX [a [b`
	返回示例：
	`(integer) 2`

EXZCARD

类别	说明
语法	`EXZCARD key`
时间复杂度	O(1)
命令描述	返回存储在Key（TairZset数据结构）中的基数（即元素的个数）。
选项	无
返回值	返回Key中的元素的数量，如果Key不存在，则返回0。

命令示例	命令示例：
	`EXZCARD testkey`
	返回示例：
	`(integer) 2`

EXZRANK

类别	说明
语法	`EXZRANK key member`
时间复杂度	O(log(N))
命令描述	返回存储在Key（TairZset数据结构）中的成员的级别，按照分数由低到高排列。 级别（或索引）从0开始计数，即分数最低的成员的级别为0。
选项	无
返回值	• 当Key中存在指定的成员，则返回成员的级别（整数）。 • 当Key或Key中的成员不存在，则返回nil。

命令示例	命令示例：
	`EXZRANK testkey b`
	返回示例：
	`(integer) 0`

EXZREVRANK

类别	说明
语法	`EXZREVRANK key member`
时间复杂度	O(log(N))
命令描述	返回存储在Key（TairZset数据结构）中的成员的级别。返回结果按照分数从高到低排列。 级别（或索引）从0开始计数，即分数最高的成员的级别为0。 **说明** 除排序规则相反外，本命令和EXZRANK用法类似。
选项	无
返回值	• 当Key中存在指定的成员，则返回成员的级别（整数）。 • 当Key或Key中的成员不存在，则返回nil。

命令示例	命令示例：
	`EXZREVRANK testkey b`
	返回示例：
	`(integer) 1`

EXZCOUNT

类别	说明
语法	`EXZCOUNT key min max`
时间复杂度	O(log(N))，其中，N为TairZset中元素的数量。 **说明** 由于采用了元素级别获取查询范围，该操作涉及的工作量和查询范围的大小即不成正比。
命令描述	返回存储在Key（TairZset数据结构）中，分数介于min和max之间的元素的个数。
选项	min、max：分别表示最小分数和最大分数，如Key中的元素采用了多维度的分数，各维度的分数之间使用井号（#）分隔。 **说明** • 在不确定Key中元素的最高分和最低分的情况下，如果想要查询Key中分数大于等于或小于等于某一特定值的元素，请将min和max分别设置为负无穷大和正无穷大。 • 默认数据范围为闭区间，如需指定开区间，则在分数范围前添加半角圆括号，例如 `(1 5` 表示返回分数大于1且小于等于5的元素。
返回值	返回分数在指定范围内的元素的数量（整数）。

命令示例	命令示例：
	`EXZCOUNT testkey (1#0#2 6#6#6`
	返回示例：
	`(integer) 1`

EXZLEXCOUNT

类别	说明
语法	`EXZLEXCOUNT key min max`
时间复杂度	O(log(N))，其中，N为TairZset中元素的数量。 **说明** 由于采用了元素级别获取查询范围，该操作涉及的工作量和查询范围的大小即不成正比。
命令描述	为确保元素按照字典序排列，若Key中所有元素分数相同，则该命令返回存储在Key，值介于min和max之间的元素的数量。 **说明** • 如果Key中元素分数不同，则返回元素未知。 • 采用 `memcmp() C` 函数逐个比对两个元素字符串中的字节。根据比对结果，由低到高排列元素。 若两个字符串包含相同的子字符串，那么字符串越长，其分值越高。
选项	min、max：分别表示成员名称的最小值和最大值（字符串形式），需指定字符的区间，例如 • 开区间：在值的前面增加半角圆括号，例如 `(a` 。 • 闭区间：在值的前面增加方括号，例如 `[a`
返回值	返回分数在指定范围内的元素的个数（整数）。

命令示例	命令示例：
	`EXZLEXCOUNT zzz [a [b`
	返回示例：
	`(integer) 2`

EXZRANKBYSCORE

类别	说明
语法	`EXZRANKBYSCORE key score`
时间复杂度	O(log(N))
命令描述	计算指定的分数在Key（TairZset数据结构）中按照分数从低到高排序的排名位置。级别（或索引）从0开始计数，即分数最低的成员的级别为0。 **说明** 若指定的分数不存在，则返回该分数在Key（TairZset数据结构）中的预计排名；若指定的分数已存在，Tair会默认将指定的分数排在已存在的分数之前。
选项	无
返回值	返回指定分数在Key中的排名。

类别	说明

命令示例	命令示例：
	`EXZRANKBYSCORE testkey 2#0#2`
	返回示例：
	`(integer) 1`

EXZREVRANKBYSCORE

类别	说明
语法	`EXZREVRANKBYSCORE key score`
时间复杂度	O(log(N))
命令描述	计算指定的分数在Key（TairZset数据结构）中按照分数从高到低排序的排名位置。级别（或索引）从0开始计数，即分数最高的成员的级别为0。 **说明** 若指定的分数不存在，则返回该分数在Key（TairZset数据结构）中的预计排名；若指定的分数已存在，Tair会默认将指定的分数排在已存在的分数之后。
选项	无
返回值	返回指定分数在Key中的排名。

命令示例	命令示例：
	`EXZREVRANKBYSCORE testkey 2#0#2`
	返回示例：
	`(integer) 1`

3.6.11. TairRoaring

TairRoaring是基于Tair引擎的Roaring Bitmap实现，本文介绍TairRoaring及其支持的命令。

TairRoaring简介

Bitmap（又名Bitset）是一种常用的数据结构，使用少量的存储空间来实现海量数据的查询优化。尽管Bitmap相比常规基于Hash结构的实现节省了大量内存空间，但是常规Bitmap对于稀疏场景下的数据存储仍不够友好，因此有了各种压缩Bitmap的实现（Comprised bitmap），Roaring Bitmap就是业界公认的一种更高效和均衡的Bitmap压缩存储的实现。

TairRoaring在此基础上完成大量优化：

- 通过2层索引和多种动态容器（Container），平衡了多种场景下性能和空间效率。
- 使用了包括SIMD instructions、Vectorization、PopCnt算法等多种工程优化，提升了计算效率，实现了高效的时空效率。
- 基于Tair提供的强大计算性能和极高的稳定性，为用户场景保驾护航。

典型场景

适用于直播、音乐、电商等行业，通过用户多维度标签，进行个性化推荐、精准营销等场景。

发布记录

注意 V2版本Breaking Change公告：

- TR.RANGEINTARRAY：V1版本的TR.RANGEINTARRAY命令名称修改为V2版本的TR.RANGE，其内容无变化。
- TR.SETRANGE：V1版本的TR.SETRANGE命令的返回值为 `OK` ，V2版本返回值为成功设置bit值为1的数量，其他内容

无变化。

1. 2021年9月13日发布TairRoaring 。V1版本
2. 2022年3月11日发布TairRoaring 版本，请将小版本升级至1.7.27及以上，详情请参见升级小版本。
   该版本优化了部分命令的实现，提升了性能。新增TR.SETBITS、TR.CLEARBITS等9个命令，向前兼容扩展2个命令，更新1个命令，更名1个命令。
   V2版本
3. 2022年4月20日发布TairRoaring 版本，请将小版本升级至1.8.1及以上，详情请参见升级小版本。
   该版本新增TR.JACCARD、TR.CONTAINS、TR.RANK命令，更新部分命令在key不存在时的返回错误（移除了`ERR key not found`），详情请参见命令列表。
   V2.2版本

最佳实践

基于TairRoaring实现人群圈选方案

前提条件

实例为Tair（Redis企业版）性能增强型，且小版本为1.7.7及以上。

> **说明** 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。集群架构暂不支持部分新增命令与参数，当前正在开发中，如有需要，请提交工单申请试用。

注意事项

操作对象为性能增强型实例中的TairRoaring数据。

命令列表

类型	命令	语法	说明	版本变更
	TR.SETBIT	`TR.SETBIT key offset value`	设置Roaring Bitmap中指定偏移量（offset）的bit值（1或者0），并返回该bit位之前的值，Roaring Bitmap的偏移量（offset）从0开始。	-（表示未更新）
	TR.SETBITS	`TR.SETBITS key offset [offset1 offset2 ... offsetN]`	设置Roaring Bitmap中指定偏移量（offset）的bit值为1，支持传入多个值。	V2新增
	TR.CLEARBITS	`TR.CLEARBITS key offset [offset1 offset2 ... offsetN]`	设置Roaring Bitmap中指定偏移量（offset）的bit值为0，若原值为0则不操作，支持传入多个值。	V2新增
	TR.SETRANGE	`TR.SETRANGE key start end`	设置Roaring Bitmap中指定区间（偏移量）的bit值为1。	V2更新，更新返回值为成功设置bit值为1的数量。
	TR.APPENDBITARRAY	`TR.APPENDBITARRAY key offset bitarray`	将由连续的0或1组成的bit数组（bitarray）插入到Roaring Bitmap中指定偏移量（offset）之后的位置，并覆盖原有数据。	V2新增
	TR.FLIPRANGE	`TR.FLIPRANGE key start end`	对Roaring Bitmap中指定区间（偏移量）的bit值执行位反转（1反转为0；0反转为1）。若指定key不存在，则自动创建目标key，并以空Roaring Bitmap对指定区间的bit值执行位反转。	V2新增

类型	命令	语法	说明	版本变更
写操作	TR.APPENDINTARRAY	`TR.APPENDINTARRAY key value [value1 value2 ... valueN]`	设置Roaring Bitmap中指定偏移量（offset）的bit值为1，支持传入多个值。 **说明** 在TairRoaring V2版本中，建议使用TR.SETBITS代替该命令。	-
	TR.SETINTARRAY	`TR.SETINTARRAY key value [value1 value2 ... valueN]`	根据传入的整形数组，创建对应的Roaring Bitmap，该命令会重置（覆盖）已存在的Roaring Bitmap对象。 **说明** 在TairRoaring V2版本中，建议使用TR.SETBITS代替该命令。	-
	TR.SETBITARRAY	`TR.SETBITARRAY key value`	根据传入的bit（由0和1组成的字符串），创建对应的Roaring Bitmap。若目标Key已存在则会重置（覆盖）原有数据。 **说明** 在TairRoaring V2版本中，建议使用TR.APPENDBITARRAY代替该命令。	-
	TR.BITOP	`TR.BITOP destkey operation key [key1 key2 ... keyN]`	对Roaring Bitmap执行集合运算操作，计算结果存储在destkey中，支持*AND*、*OR*、*XOR*、*NOT*和*DIFF*集合运算类型。 **说明** 集群架构暂不支持该命令。	-
	TR.BITOPCARD	`TR.BITOPCARD operation key [key1 key2 ... keyN]`	对Roaring Bitmap执行集合运算操作，支持*AND*、*OR*、*XOR*、*NOT*和*DIFF*集合运算类型。 **说明** 集群架构暂不支持该命令。	V2新增
	TR.OPTIMIZE	`TR.OPTIMIZE key`	优化Roaring Bitmap的存储空间。如果目标对象相对较大，且创建后以只读操作为主，可以主动执行此命令。	-
	TR.GETBIT	`TR.GETBIT key offset`	获取Roaring Bitmap中指定偏移量（offset）的bit值。	-
	TR.GETBITS	`TR.GETBITS key offset [offset1 offset2 ... offsetN]`	获取Roaring Bitmap中指定偏移量（offset）的bit值，支持查询多个值。	V2新增
	TR.BITCOUNT	`TR.BITCOUNT key [start end]`	获取Roaring Bitmap中指定区间（偏移量）bit值为1的数量。	V2更新，向前兼容。
	TR.BITPOS	`TR.BITPOS <key> <value> [count]`	获取第count个bit值为1或者0的偏移量，count为可选参数，默认为1（表示从前向后计数的第一个）。	V2更新，向前兼容。

类型	命令	语法	说明	版本变更
读操作	TR.SCAN	`TR.SCAN key start_offset [COUNT count]`	从Roaring Bitmap中指定偏移量（start_offset）开始向后扫描，返回若干（count）个bit值为1的偏移量，返回的游标（cursor）为Roaring Bitmap对应的offset。 **说明** 在迭代过程中被添加、被删除的元素的扫描结果存在不确定性，即可能被返回，也可能不会。	V2新增
	TR.RANGE	`TR.RANGE key start end`	获取Roaring Bitmap指定区间中bit值为1的偏移量。	V1的TR.RANGEINTARRAY命令，V2重命名为TR.RANGE。
	TR.RANGEBITARRAY	`TR.RANGEBITARRAY key start end`	获取Roaring Bitmap指定区间中所有bit值（0、1）组成的字符串。	V2新增
	TR.MIN	`TR.MIN key`	获取Roaring Bitmap中bit值为1的最小偏移量（首个），不存在时返回-1。	-
	TR.MAX	`TR.MAX key`	获取Roaring Bitmap中bit值为1的最大偏移量，不存在时返回-1。	-
	TR.STAT	`TR.STAT key [JSON]`	获取Roaring Bitmap的统计信息，包括各种容器的数量以及内存使用状况等信息。	V2新增
	TR.JACCARD	`TR.JACCARD key1 key2`	获取两个Roaring Bitmap之间的Jaccard相似系数，Jaccard系数值越大，Roaring Bitmap的相似度越高。 **说明** 集群架构暂不支持该命令。	V2.2新增
	TR.CONTAINS	`TR.CONTAINS key1 key2`	计算key2所对应的Roaring Bitmap是否包含key1所对应的Roaring Bitmap（即key1是否为key2的子集），若包含则返回1，否则返回0。 **说明** 集群架构暂不支持该命令。	V2.2新增
	TR.RANK	`TR.RANK key offset`	获取Roaring Bitmap中从offset为0到指定offset区间内（包含该值），bit值为1的数量。	V2.2新增
通用	DEL	`DEL key [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairRoaring数据。	-

说明

- 本文关于命令语法的定义：
  - `大写关键字`：命令关键字。
  - *斜体*：变量。
  - `[options]`：可选参数，不在括号中的参数为必选。
  - `A|B`：该组参数互斥，请进行二选一或多选一。
  - `...`：前面的内容可重复。
- 本文关于的特别约定：
  - C表示参数的数量（argc）或范围（range）。
  - M表示该种数据结构内部bit值为1的数量（例如List的node数量，Hash的field数量等）。

时间复杂度

TR.SETBIT

类别	说明
语法	`TR.SETBIT key offset value`
时间复杂度	O(1)
命令描述	设置Roaring Bitmap中指定偏移量（offset）的bit值（1或者0），并返回该bit位之前的值，Roaring Bitmap的偏移量（offset）从0开始。
选项	• Key：Key名称（TairRoaring数据结构）。 • offset：整型数字，表示待设置bit的偏移量，取值范围为0 ~ 2^32。 • value：待设置的bit值，可以设置1或者0。
返回值	• 执行成功：返回0或1，表示bit位之前的值。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.SETBIT foo 0 1`
	返回示例：
	`(integer) 0`

TR.SETBITS

类别	说明
语法	`TR.SETBITS key offset [offset1 offset2 ... offsetN]`
时间复杂度	O(C)
命令描述	设置Roaring Bitmap中指定偏移量（offset）的bit值为1，支持传入多个值。
选项	• Key：Key名称（TairRoaring数据结构）。 • offset：整型数字，表示待设置bit的偏移量，取值范围为0 ~ 2^32。

类别	说明
返回值	• 执行成功：返回更新后Roaring Bitmap中bit值为1的数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.SETBITS foo 9 10`
	返回示例：
	`(integer) 5`

TR.CLEARBITS

类别	说明
语法	`TR.CLEARBITS key offset [offset1 offset2 ... offsetN]`
时间复杂度	O(C)
命令描述	设置Roaring Bitmap中指定偏移量（offset）的bit值为0，若原值为0则不操作，支持传入多个值。
选项	• Key：Key名称（TairRoaring数据结构）。 • offset：整型数字，表示待设置bit的偏移量，取值范围为0 ~ 2^32。
返回值	• 执行成功：返回本次命令成功将bit值设置为0的数量。 • 若key不存在：返回0。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.CLEARBITS foo 9 10`
	返回示例：
	`(integer) 2`

TR.SETRANGE

类别	说明
语法	`TR.SETRANGE key start end`
时间复杂度	O(C)
命令描述	设置Roaring Bitmap中指定区间（偏移量）的bit值为1。 例如执行 `TR.SETRANGE foo 1 3` ，将创建foo为"0111"。
选项	• Key：Key名称（TairRoaring数据结构）。 • start：起始偏移量（包含该值），取值范围为0 ~ 2^32。 • end：结束偏移量（包含该值），取值范围为0 ~ 2^32。
返回值	• 执行成功：返回本次命令成功将bit值设置为1的数量。 • 其他情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`TR.SETRANGE foo 1 3`
	返回示例：
	`(integer) 3`

TR.APPENDBITARRAY

类别	说明
语法	`TR.APPENDBITARRAY key offset bitarray`
时间复杂度	O(C)
命令描述	将由连续的0或1组成的bit数组（bitarray）插入到Roaring Bitmap中指定偏移量（offset）之后的位置，并覆盖原有数据。
选项	• Key：Key名称（TairRoaring数据结构）。 • offset：指定的起始偏移量（不包含该值），取值范围为-1 ~ 2^32。 • bitarray：待添加的bit数组，将覆盖原有数据，由连续的0或1，取值范围为0 ~ 2^32。 **说明** 指定的offset与添加的bitarray的总长度不能超过2^32，否则会操作失败。
返回值	• 执行成功：返回本次命令成功将bit值设置为1的数量。 • 其他情况返回相应的异常信息。

示例	提前执行`TR.SETBITS foo 0`。
	命令示例：
	`TR.APPENDBITARRAY foo 1 1101`
	返回示例：
	`(integer) 3`
	此时，Roaring Bitmap foo为“101101”。

TR.FLIPRANGE

类别	说明
语法	`TR.FLIPRANGE key start end`
时间复杂度	O(C)
命令描述	对Roaring Bitmap中指定区间（偏移量）的bit值执行位反转（1反转为0；0反转为1）。若指定key不存在，则自动创建目标key，并以空Roaring Bitmap对指定区间的bit值执行位反转。
选项	• Key：Key名称（TairRoaring数据结构）。 • start：起始偏移量（包含该值），取值范围为0 ~ 2^32。 • end：结束偏移量（包含该值），取值范围为0 ~ 2^32。

类别	说明
返回值	• 执行成功：返回本次命令成功将bit值设置为1的数量。 • 若key不存在：自动创建目标key，并以空Roaring Bitmap对指定区间的bit值执行位反转，返回本次命令成功将bit值设置为1的数量。 • 其他情况返回相应的异常信息。

示例	提前执行 `TR.SETBITS foo 0 2 3 5` 。
	命令示例：
	`TR.FLIPRANGE foo 0 5`
	返回示例：
	`(integer) 2`
	此时，Roaring Bitmap foo为“01001”。

TR.APPENDINTARRAY

类别	说明
语法	`TR.APPENDINTARRAY key value [value1 value2 ... valueN]`
时间复杂度	O(C)
命令描述	设置Roaring Bitmap中指定偏移量（offset）的bit值为1，支持传入多个值。 **说明** 在TairRoaring V2版本中，建议使用TR.SETBITS代替该命令。
选项	• Key：Key名称（TairRoaring数据结构）。 • value：整形数字，表示待设置的bit位，取值范围为0 ~ 4294967296。
返回值	• 执行成功：返回OK。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.APPENDINTARRAY foo 9 10`
	返回示例：
	`OK`

TR.SETINTARRAY

类别	说明
语法	`TR.SETINTARRAY key value [value1 value2 ... valueN]`
时间复杂度	O(C)
命令描述	根据传入的整形数组来设置对应的Roaring Bitmap，若目标Key已存在则会重置（覆盖）原有数据。 **说明** 在TairRoaring V2版本中，建议使用TR.SETBITS代替该命令。

类别	说明
选项	• Key：Key名称（TairRoaring数据结构）。 • value：整形数字，表示待设置的bit位。
返回值	• 执行成功：返回OK。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.SETINTARRAY foo 2 4 5 6`
	返回示例：
	`OK`

TR.SETBITARRAY

类别	说明
语法	`TR.SETBITARRAY key value`
时间复杂度	O(C)
命令描述	根据传入的bit（由0和1组成的字符串），创建对应的Roaring Bitmap。若目标Key已存在则会重置（覆盖）原有数据。 **说明** 在TairRoaring V2版本中，建议使用TR.APPENDBITARRAY代替该命令。
选项	• key：Key名称（TairRoaring数据结构）。 • value：由0和1构成的字符串，即需要设置的bit数组。
返回值	• 执行成功：返回 `OK` 。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`tr.setbitarray foo 10101001`
	返回示例：
	`OK`

TR.BITOP

类别	说明
语法	`TR.BITOP destkey operation key [key1 key2 ... keyN]`
时间复杂度	O(C * M)

类别	说明
命令描述	对Roaring Bitmap执行集合运算操作，计算结果存储在destkey中，支持*AND*、*OR*、*XOR*、*NOT*和*DIFF*集合运算类型。 **说明** 集群架构暂不支持该命令。
选项	• key：Key名称（TairRoaring数据结构），可传入多个Key。 • operation：集合运算类型，取值：*AND*（表示与）、*OR*（表示或）、*XOR*（表示异或）、*NOT*（表示非）、*DIFF*（表示差）。 **说明** ◦ *NOT*仅支持操作1个对象。 ◦ *DIFF*仅支持计算2个对象的差集，请注意计算差集对象的运算顺序，例如`TR.BITOP result DIFF key1 key2`是计算key1关于key2的差集（key1 - key2）。 • destkey：集合运算结果所存储的目标Key（TairRoaring数据结构）。
返回值	• 执行成功：返回操作运算结果中bit值为1的数量，格式为Integer（整数）。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.BITOP result OR foo bar`
	返回示例：
	`(integer) 6`

TR.BITOPCARD

类别	说明
语法	`TR.BITOPCARD operation key [key1 key2 ... keyN]`
时间复杂度	O(C * M)
命令描述	对Roaring Bitmap执行集合运算操作，支持*AND*、*OR*、*XOR*、*NOT*和*DIFF*集合运算类型。 **说明** 集群架构暂不支持该命令。
选项	• key：Key名称（TairRoaring数据结构），可传入多个Key。 • operation：集合运算类型，取值：*AND*（表示与）、*OR*（表示或）、*XOR*（表示异或）、*NOT*（表示非）、*DIFF*（表示差）。 **说明** ◦ *NOT*仅支持操作1个对象。 ◦ *DIFF*仅支持计算2个对象的差集，请注意计算差集对象的运算顺序，例如`TR.BITOP result DIFF key1 key2`是计算key1关于key2的差集（key1 - key2）。

类别	说明
返回值	• 执行成功：返回操作运算结果中bit值为1的数量，格式为Integer（整数）。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.BITOPCARD NOT foo`
	返回示例：
	`(integer) 2`

TR.OPTIMIZE

类别	说明
语法	`TR.OPTIMIZE key`
时间复杂度	O(M)
命令描述	优化Roaring Bitmap的存储空间。如果目标对象相对较大，且创建后以只读操作为主，可以主动执行此命令。
选项	• Key：Key名称（TairRoaring数据结构）。
返回值	• 执行成功：返回更新后Roaring Bitmap中bit值为1的数量。 • 若key不存在：返回 `nil` 。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.OPTIMIZE foo`
	返回示例：
	`OK`

TR.GETBIT

类别	说明
语法	`TR.GETBIT key offset`
时间复杂度	O(1)
命令描述	获取Roaring Bitmap中指定偏移量（offset）的bit值。
选项	• Key：Key名称（TairRoaring数据结构）。 • offset：待查询的偏移量。
返回值	• 执行成功：返回0或1，表示bit位的值。 • 其他情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`TR.GETBIT foo 0`
	返回示例：
	`(integer) 1`

TR.GETBITS

类别	说明
语法	`TR.GETBITS key offset [offset1 offset2 ... offsetN]`
时间复杂度	O(C)
命令描述	获取Roaring Bitmap中指定偏移量（offset）的bit值，支持查询多个值。
选项	• Key：Key名称（TairRoaring数据结构）。 • offset：待查询的偏移量。
返回值	• 执行成功：返回对应bit的值。 • 若key不存在：返回空数组。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.GETBITS foo 3 4 6 8`
	返回示例：
	`1) (integer) 1`
	`2) (integer) 1`
	`3) (integer) 1`
	`4) (integer) 0`

TR.BITCOUNT

类别	说明
语法	`TR.BITCOUNT key [start end]`
时间复杂度	O(M)
命令描述	获取Roaring Bitmap中指定区间（偏移量）bit值为1的数量。
选项	• Key：Key名称（TairRoaring数据结构）。 • start：起始偏移量（包含该值），取值范围为0 ~ 2^32。 • end：结束偏移量（包含该值），取值范围为0 ~ 2^32。
返回值	• 执行成功：返回Roaring Bitmap中值为1的bit位数量，格式为Integer（整数）。 • 若key不存在：返回0。 • 其他情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`TR.BITCOUNT foo 4 9`
	返回示例：
	`(integer) 3`

TR.BITPOS

类别	说明
语法	`TR.BITPOS <key> <value> [count]`
时间复杂度	O(C)
命令描述	获取第count个bit值为1或者0的偏移量，count为可选参数，默认为1（表示从前向后计数的第一个）。
选项	• Key：Key名称（TairRoaring数据结构）。 • value：待查找bit值（1或者0）。 • count：查找第几位，负数表示从末尾向前计数。
返回值	• 执行成功：返回目标bit的偏移量（offset）。 • 若key不存在：返回-1。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.BITPOS foo 1 -1`
	返回示例：
	`(integer) 6`

TR.SCAN

类别	说明
语法	`TR.SCAN key start_offset [COUNT count]`
时间复杂度	O(C)
命令描述	从Roaring Bitmap中指定偏移量（start_offset）开始向后扫描，返回若干（count）个bit值为1的偏移量，返回的游标（cursor）为Roaring Bitmap对应的offset。 **说明** 在迭代过程中被添加、被删除的元素的扫描结果存在不确定性，即可能被返回，也可能不会。
选项	• Key：Key名称（TairRoaring数据结构）。 • start_offset：起始偏移量（包含该值）。 • COUNT：查询的数量（默认为10）。

类别	说明

返回值	• 执行成功，返回具有两个元素的数组：
	  ◦ 第一个元素：下次查询的start_offset，若该key已扫描完成，则返回0。
	  ◦ 第二个元素：本次查询的目标偏移量。
	  说明 若key不存在，返回0与空元素组成的数组。
	• 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.SCAN foo 0 COUNT 2`
	返回示例：
	`1) (integer) 3`
	`2) 1) (integer) 0`
	`2) (integer) 2`

TR.RANGE

类别	说明
语法	`TR.RANGE key start end`
时间复杂度	O(C)
命令描述	获取Roaring Bitmap指定区间中bit值为1的偏移量。
选项	• Key：Key名称（TairRoaring数据结构）。 • start：起始的偏移量（包含该值）。 • end：结束的偏移量（包含该值）。
返回值	• 执行成功：返回bit值为1的偏移量。 • 若key不存在：返回空数组。 • 其他情况返回相应的异常信息。

示例	提前执行 `TR.SETBITS foo 0 2 3 5` 。
	命令示例：
	`TR.RANGE foo 0 5`
	返回示例：
	`1) (integer) 0`
	`2) (integer) 2`
	`3) (integer) 3`
	`4) (integer) 5`

TR.RANGEBITARRAY

类别	说明
语法	`TR.RANGEBITARRAY key start end`

类别	说明
时间复杂度	O(C)
命令描述	获取Roaring Bitmap指定区间中所有bit值（0、1）组成的字符串。
选项	• Key：Key名称（TairRoaring数据结构）。 • start：起始的偏移量（包含该值）。 • end：结束的偏移量（包含该值）。
返回值	• 执行成功：返回bit值为1的偏移量。 • 若key不存在：返回 `nil` 。 • 其他情况返回相应的异常信息。

示例	提前执行 `TR.SETBITS foo 0 2 3 5` 。
	命令示例：
	`TR.RANGEBITARRAY foo 0 5`
	返回示例：
	`"101101"`

TR.MIN

类别	说明
语法	`TR.MIN key`
时间复杂度	O(1)
命令描述	获取Roaring Bitmap中bit值为1的最小偏移量（首个），不存在时返回-1。
选项	• Key：Key名称（TairRoaring数据结构）。

返回值	• 执行成功：
	  ◦ 返回首个bit值为1的偏移量，格式为Integer（整数）。
	  ◦ 返回-1，表示key不存在或者该Roaring Bitmap中不存在值为1的bit。
	• 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.MIN foo`
	返回示例：
	`4`

TR.MAX

类别	说明
语法	`TR.MAX key`
时间复杂度	O(1)

类别	说明
命令描述	获取Roaring Bitmap中bit值为1的最大偏移量，不存在时返回-1。
选项	• Key：Key名称（TairRoaring数据结构）。

返回值	• 执行成功：
	  ◦ 返回最后一个bit值为1的偏移量，格式为Integer（整数）。
	  ◦ 返回-1，表示key不存在或者该Roaring Bitmap中不存在值为1的bit。
	• 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.MAX foo`
	返回示例：
	`6`

TR.STAT

类别	说明
语法	`TR.STAT key [JSON]`
时间复杂度	O(M)
命令描述	获取Roaring Bitmap的统计信息，包括各种容器的数量以及内存使用状况等信息。
选项	• Key：Key名称（TairRoaring数据结构）。 • JSON：若指定JSON，则以JSON格式返回统计信息。

返回值	• 执行成功：返回Redis的统计信息（bulk string），说明如下。
	`"{\"cardinality\":3,` # 元素总数量
	`\"number_of_containers\":1,` # TairRoaring容器总数（Roaring Bitmap概念）
	`\"max_value\":6,` # 最大元素值
	`\"min_value\":3,` # 最小元素值
	`\"sum_value\":13,`
	`\"array_container\":{` # array容器数量（Roaring Bitmap概念）
	`\"number_of_containers\":1,`
	`\"container_cardinality\":3,`
	`\"container_allocated_bytes\":6},`
	`\"bitset_container\":{` # bitset容器数量（Roaring Bitmap概念）
	`\"number_of_containers\":0,`
	`\"container_cardinality\":0,`
	`\"container_allocated_bytes\":0},`
	`\"run_container\":{` # RLE容器数量（Roaring Bitmap概念）
	`\"number_of_containers\":0,`
	`\"container_cardinality\":0,`
	`\"container_allocated_bytes\":0}}"`
	• 若key不存在：返回 `nil` 。
	• 其他情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`TR.STAT foo JSON`
	
	返回示例：
	`"{\"cardinality\":4,\"number_of_containers\":1,\"max_value\":5,\"min_value\":0,\"sum_value\":10,\"array_container\":{\"number_of_containers\":1,\"container_cardinality\":4,\"container_allocated_bytes\":8},\"bitset_container\":{\"number_of_containers\":0,\"container_cardinality\":0,\"container_allocated_bytes\":0},\"run_container\":{\"number_of_containers\":0,\"container_cardinality\":0,\"container_allocated_bytes\":0}}"`

TR.JACCARD

类别	说明
语法	`TR.JACCARD key1 key2`
时间复杂度	O(M)
命令描述	获取两个Roaring Bitmap之间的Jaccard相似系数，Jaccard系数值越大，Roaring Bitmap的相似度越高。 **说明** 集群架构暂不支持该命令。
选项	• Key：Key名称（TairRoaring数据结构）。
返回值	• 执行成功：返回Jaccard相似系数（Double类型）。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TR.JACCARD foo1 foo2`
	
	返回示例：
	`"0.20000000000000001"`

TR.CONTAINS

类别	说明
语法	`TR.CONTAINS key1 key2`
时间复杂度	O(M)
命令描述	计算key2所对应的Roaring Bitmap是否包含key1所对应的Roaring Bitmap（即key1是否为key2的子集），若包含则返回1，否则返回0。 **说明** 集群架构暂不支持该命令。

类别	说明
选项	• Key：Key名称（TairRoaring数据结构）。

返回值	• 执行成功：
	  ◦ 返回1，表示key2包含key1，即key1是key2的子集。
	  ◦ 返回0，表示key2不包含key1，即key1不是key2的子集。
	• 其他情况返回相应的异常信息。

示例	提前执行 `TR.SETBITS fooM 1 2 3 10` 、 `TR.SETBITS foom 1 2` 命令。
	命令示例：
	`TR.CONTAINS foom fooM`
	返回示例：
	`(integer) 1`

TR.RANK

类别	说明
语法	`TR.RANK key offset`
时间复杂度	O(M)
命令描述	获取Roaring Bitmap中从offset为0到指定offset区间内（包含该值），bit值为1的数量。
选项	• Key：Key名称（TairRoaring数据结构）。 • offset：指定bit的offset位，取值为INT（整形数字）。
返回值	• 执行成功：目标区间bit值为1的数量。 • 其他情况返回相应的异常信息。

示例	提前执行 `TR.SETBITS fooM 1 2 3 10` 。
	命令示例：
	`TR.RANK fooM 10`
	返回示例：
	`(integer) 4`

异常返回值说明

错误信息	说明
`WRONGTYPE Operation against a key holding the wrong kind of value`	对象类型错误：Key不是TairRoaring对象。
`ERR bad arguments, must be unsigned 32-bit integer`	参数类型错误：无法按照32-bit整形进行转换。

错误信息	说明
`ERR invalid arguments, maybe out of range or illegal`	参数非法： • 非32-bit整形的offset不符合规则。 • 参数的[start,end]不符合规则。 • 参数超过Roaring Bitmap的元素个数。
`ERR key already exist`	Roaring Bitmap对象已存在，且不支持覆盖。 **说明** V2.2版之后将不会产生该报错。
`ERR key not found`	Roaring Bitmap对象不存在，不支持操作。 **说明** V2.2版之后将不会产生该报错。

3.6.12. TairSearch

TairSearch是基于Redis module全自研（不基于Lucene等开源搜索库）的全文搜索数据结构，采用和Elasticsearch相似（ES-LIKE）的查询语法。

TairSearch简介

与Elasticsearch相比，TairSearch具有如下主要特点：

- 全部数据和索引均在内存中，具有更高的吞吐性能、更低的读写延迟。
- 支持文档的局部索引更新，包括追加字段、更新字段、删除字段以及字段自增等。
- 实时索引，写入即可见。

前提条件

实例为Tair（Redis企业版）性能增强型，且小版本为1.7.27及以上。

说明 最新小版本将提供更丰富的功能与稳定的服务，建议将实例的小版本升级到最新，具体操作请参见升级小版本。若实例为集群架构或读写分离架构，请将代理节点的小版本也升级到最新，避免代理节点无法识别部分命令。

注意事项

- 操作对象为性能增强型实例中的TairSearch数据。
- 为节省内存，推荐使用如下方法：
  - 创建索引（index）时请将文档中需要创建（反向）索引的字段设置为索引字段（将字段的index设置为true），其余字段的index设置为false。
  - 使用_source参数中的include与exclude机制剔除源文档中不需要的字段（field），保存需要的信息。
- 避免在单个index中插入过多的文档，推荐将文档存入多个不同的index中，建议单个index的文档数在500万以下，可有效规避（集群）实例发生数据倾斜，均衡读写流量，避免造成Bigkey与Hotkey。

命令列表

全文检索命令

命令	语法	说明
TFT.CREATEINDEX	`TFT.CREATEINDEX index mappings`	创建索引（index）并添加映射（mappings），mappings语法类似ES语法。在添加索引文档前，必须先创建索引（index）。
TFT.UPDATEINDEX	`TFT.UPDATEINDEX index mappings`	向指定的index中新增properties字段，不能与原有字段冲突，否则会新增失败。
TFT.GETINDEX	`TFT.GETINDEX index`	获取index的映射内容。

命令	语法	说明
TFT.ADDDOC	`TFT.ADDDOC index document [WITH_ID doc_id]`	向index中插入一个文档（document），可通过WITH_ID指定该文档在索引内的唯一ID（doc_id），若doc_id已存在，则更新并覆盖原文档。若不指定WITH_ID（默认），则自动生成doc_id。
TFT.MADDDOC	`TFT.MADDDOC index document doc_id [document1 doc_id1] ...`	向index中插入多个文档（document），每个文档必须指定文档ID（doc_id）。
TFT.UPDATEDOCFIELD	`TFT.UPDATEDOCFIELD index doc_id document`	更新index中doc_id指定的文档，若更新的字段为mapping指定的索引字段时，该字段更新的内容需与mapping指定的类型一致；若非索引字段，支持更新任意字段类型的内容。 **说明** 若更新的字段已存在，则更新原文档，若字段不存在，则新增该字段。若指定的文档不存在，该命令支持自动创建文档，此时效果等同于TFT.ADDDOC。
TFT.DELDOCFIELD	`TFT.DELDOCFIELD index doc_id field [field1 field2 ...]`	删除index中doc_id指定文档的指定字段，若该字段为索引字段，会同时在索引中删除该字段的信息。 **说明** 若指定的字段不存在（例如被_source过滤的字段），则操作失败。
TFT.INCRLONGDOCFIELD	`TFT.INCRLONGDOCFIELD index doc_id field increment`	向index中doc_id指定文档的指定字段增加整数值（increment），支持指定increment为负数，支持指定的字段类型为long或int类型。 **说明** 若指定的文档不存在，该命令支持自动创建文档，初始化字段的值为0，并增加指定的increment。若指定的字段不存在（例如被_source过滤的字段），则操作失败。
TFT.INCRFLOATDOCFIELD	`TFT.INCRFLOATDOCFIELD index doc_id field increment`	向index中doc_id指定文档的指定字段增加浮点数值（increment），支持指定increment为负数，支持指定的字段类型为int、long或double类型。 **说明** 若指定的文档不存在，该命令支持自动创建文档，初始化字段的值为0，并增加指定的increment。若指定的字段不存在（例如被_source过滤的字段），则操作失败。
TFT.GETDOC	`TFT.GETDOC index doc_id`	获取index中指定doc_id的文档内容。
TFT.EXISTS	`TFT.EXISTS index doc_id`	查询index中指定doc_id的文档是否存在。
TFT.DOCNUM	`TFT.DOCNUM index`	获取index中的文档数量。
TFT.SCANDOCID	`TFT.SCANDOCID index cursor [MATCH *value*] [COUNT count]`	获取index中所有的doc_id。
TFT.DELDOC	`TFT.DELDOC index doc_id [doc_id] ...`	删除index中doc_id指定的文档，支持指定多个doc_id。
TFT.DELALL	`TFT.DELALL index`	删除index中所有文档，但不会删除index。
TFT.SEARCH	`TFT.SEARCH index query`	根据query语句搜索index的文档，query语法类似ES语法。
DEL	`DEL key [key ...]`	使用原生Redis的DEL命令可以删除一条或多条TairSearch数据。

自动补齐命令

命令	语法	说明
TFT.ADDSUG	`TFT.ADDSUG index text weight [text weight] ...`	在指定索引中，添加自动补全的文本及对应权重，支持添加多个文本。
TFT.DELSUG	`TFT.DELSUG index text [text] ...`	在指定索引中，删除自动补全的文本，支持删除多个文本。
TFT.SUGNUM	`TFT.SUGNUM index`	获取指定索引中自动补全文本的数量。
TFT.GETSUG	`TFT.GETSUG index prefix [MAX_COUNT count] [FUZZY]`	根据指定前缀，查询匹配的自动补全文本，将优先返回权重比较高的text。
TFT.GETALLSUGS	`TFT.GETALLSUGS index`	获取指定索引的全量自动补全文本。

说明 本文的命令语法定义如下：

- `大写关键字`：命令关键字。
- *斜体*：变量。
- `[options]`：可选参数，不在括号中的参数为必选。
- `A|B`：该组参数互斥，请进行二选一或多选一。
- `...`：前面的内容可重复。

TFT.CREATEINDEX

类别	说明
语法	`TFT.CREATEINDEX index mappings`
命令描述	创建索引（index）并添加映射（mappings），mappings语法类似ES语法。在添加索引文档前，必须先创建索引（index）。

类别	说明

选项	• index：待创建的索引名称。
	• mappings：映射内容，支持的语法如下。
	  ○ dynamic：映射模式，取值如下。
	  ■ strict：严格映射，只有在properties中配置过的field才会创建映射，且文档中出现未映射的field无法成功写入并报错。若未指定该参数，则默认为非严格映射模式。
	  ○ enabled：强制检查写入文档的field类型与property中指定的类型是否一致，若不一致则写入失败，取值为true（默认，表示会检查）和false，该参数仅对非索引字段（index为true）生效，索引字段会强制检查字段类型。
	  ○ _source：设置存储原始文档，该参数不会影响对应field索引的创建，取值如下。
	  ■ enabled：是否存储原始文档，取值如下。
	  ■ true（默认）：存储原始文档，默认存储所有字段。
	可指定excludes（不需要存储的field）和includes（需要存储的field），支持配置通配符模式（WildCard）的field。若某一个field同时出现在excludes和includes时，则excludes的优先级大于includes，表示该field最终不会存储在原始文档中。示例如下。
	示例一：`"_source":{"enabled":true}`（默认），表示存储所有字段，相同于`"_source":{"enabled":true,"excludes":[],"includes":[]}`。
	示例二：`"_source":{"enabled":true,"excludes":[],"includes":["id","name"]}`，表示仅存储“id”、“name”字段。
	  ■ false：不存储原始文档。
	  ○ properties：映射的field集合，每个field可设置如下属性如下。
	  ■ index：是否将该field加入索引，取值：true（默认）、false。
	  ■ boost：该field的计分权重，默认为1，范围为正浮点数，Tair将自动计算出各个field的相对计算分。
	  ■ type：field的数据类型，取值如下。
	  ■ keyword：不可被拆分字符串。可指定如下参数：
	  ■ ignore_above：指定keyword的字符串长度上限，默认为128，例如`"ignore_above":128`。
	  ■ text：字符串，且可通过分词器解析，存入索引。可指定如下参数：
	  ■ analyzer：解析text存入索引的分词器，支持standard（默认，英文分词器）、chinese、arabic、cjk、czech、brazilian、german、greek、persian、french、dutch、russian和jieba（推荐的中文分分词，效果比chinese好）。例如`"analyzer":"jieba"`，表示使用中文分词器。
	  ■ search_analyzer：指定搜索时使用的分词器，支持的分词器类型与analyzer相同，默认为analyzer的设置。
	  ■ long：long数据类型，您可以将时间点转为Unix时间戳，存入该数据类型中。
	  ■ integer：int数据类型。
	  ■ double：double数据类型。

返回值	• 执行成功：返回OK。 • 其他情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`TFT.CREATEINDEX idx:product '{"mappings":{"_source":{"enabled":true},"properties":{"product_id":{"type":"keyword","ignore_above":128},"product_name":{"type":"text"},"product_title":{"type":"text","analyzer":"jieba"},"price":{"type":"double"}}}}'`
	`# 创建商品ID（product_id）为keyword数据类型，并设置上限为128个字符。`
	`# 创建商品名称（product_name）为text数据类型。`
	`# 创建商品标题（product_title）为text数据类型，并设置分词器为中文。`
	`# 创建价格（price）为double数据类型。`
	
	返回示例：
	`OK`

TFT.UPDATEINDEX

类别	说明
语法	`TFT.UPDATEINDEX index mappings`
命令描述	向指定的index中新增properties字段，不能与原有字段冲突，否则会新增失败。
选项	• index：待操作的索引名称。 • mappings：映射内容，无需输入dynamic、_source等信息，仅输入新增的properties字段，语法请参见TFT.CREATEINDEX。
返回值	• 执行成功：返回OK。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.UPDATEINDEX idx:product '{"mappings":{"properties":{"product_group":{"type":"text","analyzer":"chinese"}}}}'`
	
	返回示例：
	`OK`

TFT.GETINDEX

类别	说明
语法	`TFT.GETINDEX index`
命令描述	获取index的映射内容。
选项	• index：待操作的索引名称。
返回值	• 执行成功：返回索引的映射内容，格式为JSON。 • 其他情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`TFT.GETINDEX idx:product`
	
	返回示例：
	`{"idx:product0310":{"mappings":{"_source":{"enabled":true,"excludes":[],"includes":["product_id"]},"dynamic":"false","properties":{"price":{"boost":1.0,"enabled":true,"ignore_above":-1,"index":true,"similarity":"classic","type":"double"},"product_id":{"boost":1.0,"enabled":true,"ignore_above":128,"index":true,"similarity":"classic","type":"keyword"},"product_name":{"boost":1.0,"enabled":true,"ignore_above":-1,"index":true,"similarity":"classic","type":"text"},"product_title":{"analyzer":"chinese","boost":1.0,"enabled":true,"ignore_above":-1,"index":true,"similarity":"classic","type":"text"}}}}}`

TFT.ADDDOC

类别	说明
语法	`TFT.ADDDOC index document [WITH_ID doc_id]`
命令描述	向index中插入一个文档（document），可通过WITH_ID指定该文档在索引内的唯一ID（doc_id），若doc_id已存在，则更新并覆盖原文档。若不指定WITH_ID（默认），则自动生成doc_id。
选项	• index：待操作的索引名称。 • document：插入的文档，JSON格式。 **说明** 若index通过_source的includes参数配置了仅保存部分字段，则在插入或更新文档时仅保存includes中配置的字段。 • WITH_ID doc_id：是否指定文档ID，如需指定文档ID需输入doc_id值，doc_id的格式为任意字符串。
返回值	• 执行成功：返回文档ID，格式为JSON。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.ADDDOC idx:product '{"product_id":"product test"}' WITH_ID 00001`
	
	返回示例：
	`"{\"_id\":\"00001\"}"`

TFT.MADDDOC

类别	说明
语法	`TFT.MADDDOC index document doc_id [document1 doc_id1] ...`
命令描述	向index中插入多个文档（document），每个文档必须指定文档ID（doc_id）。

类别	说明
选项	• index：待操作的索引名称。 • document：插入的文档，JSON格式。 **说明** 若index通过_source的includes参数配置了仅保存部分字段，则在插入或更新文档时仅保存includes中配置的字段。 • doc_id：指定文档ID，doc_id的格式为任意字符串。
返回值	• 执行成功：返回OK。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.MADDDOC idx:product '{"product_id":"test1"}' 00011 '{"product_id":"test2"}' 00012`
	返回示例：
	`OK`

TFT.UPDATEDOCFIELD

类别	说明
语法	`TFT.UPDATEDOCFIELD index doc_id document`
命令描述	更新index中doc_id指定的文档，若更新的字段为mapping指定的索引字段时，该字段更新的内容需与mapping指定的类型一致；若非索引字段，支持更新任意字段类型的内容。 **说明** 若更新的字段已存在，则更新原文档，若字段不存在，则新增该字段。若指定的文档不存在，该命令支持自动创建文档，此时效果等同于TFT.ADDDOC。
选项	• index：待操作的索引名称。 • doc_id：指定文档ID。 • document：更新的文档内容，JSON格式。 **说明** 若index通过_source的includes参数配置了仅保存部分字段，则在插入或更新文档时仅保存includes中配置的字段。
返回值	• 执行成功：返回OK。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.UPDATEDOCFIELD idx:product 00011 '{"product_id":"test8","product_group":"BOOK"}'`
	返回示例：
	`OK`

TFT.DELDOCFIELD

类别	说明
语法	`TFT.DELDOCFIELD index doc_id field [field1 field2 ...]`
命令描述	删除index中doc_id指定文档的指定字段，若该字段为索引字段，会同时在索引中删除该字段的信息。 **说明** 若指定的字段不存在（例如被_source过滤的字段），则操作失败。
选项	• index：待操作的索引名称。 • doc_id：指定文档ID。 • field：待删除的字段。
返回值	• 执行成功：返回成功删除的字段数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.DELDOCFIELD idx:product 00011 product_group`
	返回示例：
	`(integer) 1`

TFT.INCRLONGDOCFIELD

类别	说明
语法	`TFT.INCRLONGDOCFIELD index doc_id field increment`
命令描述	向index中doc_id指定文档的指定字段增加整数值（increment），支持指定increment为负数，支持指定的字段类型为long或int类型。 **说明** 若指定的文档不存在，该命令支持自动创建文档，初始化字段的值为0，并增加指定的increment。若指定的字段不存在（例如被_source过滤的字段），则操作失败。
选项	• index：待操作的索引名称。 • doc_id：指定文档ID。 • field：待操作的字段，支持的字段类型为long或int类型。 • increment：待增加操作的值，可以指定该值为负数实现相减，数据类型为整型（int）。
返回值	• 执行成功：返回执行操作后字段的数值。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.INCRLONGDOCFIELD idx:product 00011 stock 100`
	返回示例：
	`(integer)100`

TFT.INCRFLOATDOCFIELD

类别	说明
语法	`TFT.INCRFLOATDOCFIELD index doc_id field increment`
命令描述	向index中doc_id指定文档的指定字段增加浮点数值（increment），支持指定increment为负数，支持指定的字段类型为int、long或double类型。 **说明** 若指定的文档不存在，该命令支持自动创建文档，初始化字段的值为0，并增加指定的increment。若指定的字段不存在（例如被_source过滤的字段），则操作失败。
选项	• index：待操作的索引名称。 • doc_id：指定文档ID。 • field：待操作的字段，支持的字段类型为long、int或double类型。 • increment：待增加操作的值，可以指定该值为负数实现相减，数据类型为双精度浮点型（double）。
返回值	• 执行成功：返回执行操作后字段的数值。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.INCRFLOATDOCFIELD idx:product 00011 stock 299.6`
	返回示例：
	`"299.6"`

TFT.GETDOC

类别	说明
语法	`TFT.GETDOC index doc_id`
命令描述	获取index中指定doc_id的文档内容。
选项	• index：待查询的索引名称。 • doc_id：指定文档ID。
返回值	• 执行成功：返回文档ID和文档内容。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.GETDOC idx:product 00011`
	返回示例：
	`"{\"_id\":\"00011\",\"_source\":{\"product_id\":\"test8\"}}"`

TFT.EXISTS

类别	说明
语法	`TFT.EXISTS index doc_id`
命令描述	查询index中指定doc_id的文档是否存在。

类别	说明
选项	• index：待查询的索引名称。 • doc_id：指定文档ID。

返回值	• 执行成功：
	  ◦ 若文档存在，返回1。
	  ◦ 若索引或文档不存在，返回0。
	• 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.EXISTS idx:product 00011`
	返回示例：
	`(integer) 1`

TFT.DOCNUM

类别	说明
语法	`TFT.DOCNUM index`
命令描述	获取index中的文档数量。
选项	• index：待查询的索引名称。
返回值	• 执行成功：索引的文档数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.DOCNUM idx:product`
	返回示例：
	`(integer) 3`

TFT.SCANDOCID

类别	说明
语法	`TFT.SCANDOCID index cursor [MATCH *value*] [COUNT count]`
命令描述	获取index中所有的doc_id。
选项	• index：待查询的索引名称。 • cursor：指定本次扫描的游标。 • MATCH *value*：模式匹配，value为待匹配的值，例如 `MATCH *redis*` 。 • COUNT count：指定本次扫描的最大数量，默认为100。

类别	说明

返回值	• 执行成功：返回一个数组。
	  ◦ 第一个元素：下次查询的cursor，若该key已扫描完成，则返回0。
	  ◦ 第二个元素：本次查询的doc_id。
	• 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.SCANDOCID idx:product 0 COUNT 3`
	返回示例：
	`1) "0"`
	`2) 1) "00001"`
	`2) "00011"`
	`3) "00012"`

TFT.DELDOC

类别	说明
语法	`TFT.DELDOC index doc_id [doc_id] ...`
命令描述	删除index中doc_id指定的文档，支持指定多个doc_id。
选项	• index：待查询的索引名称。 • doc_id：指定文档ID。
返回值	• 执行成功：返回成功删除的文档数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.DELDOC idx:product 00011 00014`
	返回示例：
	`"1"  # 本示例中ID"00014"文档不存在，故仅删除ID"00011"文档，返回1。`

TFT.DELALL

类别	说明
语法	`TFT.DELALL index`
命令描述	删除index中所有文档，但不会删除index。
选项	• index：待查询的索引名称。 • doc_id：指定文档ID。
返回值	• 执行成功：返回OK。 • 其他情况返回相应的异常信息。

类别	说明

示例	命令示例：
	`TFT.DELALL idx:product`
	返回示例：
	`OK`

TFT.SEARCH

类别	说明
语法	`TFT.SEARCH index query`
命令描述	根据query语句搜索index的文档，query语法类似ES语法。

	• index：待查询的索引名称。
	• query：类似ES语法的DLS语句，支持如下字段。
	  ◦ query：查询子句，支持如下语法。
	  ▪ match：基础的匹配查询，支持如下参数。
	在不需要对文档进行分词拆分或模糊匹配的场景下，推荐使用term、prefix等命令，以提高查询效率。
	  ▪ query：查询的文档内容。
	说明
	▪ 未开启模糊匹配（fuzziness）时，query中的文档会被指定分词器（analyzer）拆分成多个词根，同时您可以指定operator（不同词根之间的关系）、minimum_should_match（至少需要匹配多少个词根）参数。
	▪ 开启了模糊匹配（fuzziness）时，上述参数将无效，您可以指定prefix_length参数。
	  ▪ analyzer：指定match查询语句的分词器，支持standard（默认，英文分词器）、chinese、arabic、cjk、czech、brazilian、german、greek、persian、french、dutch、russian和jieba（中分分词）。
	  ▪ minimum_should_match：查询语句会通过分词器拆分成多个词根，您可以通过本参数指定至少需要匹配多少个词根。该参数在未开启模糊匹配且operator为OR时生效。
	  ▪ operator：指定通过分词器拆分的词根之间的关系，取值为AND和OR（默认）。例如 `'{"query":{"match":{"f0":{"query":"redis cluster","operator":"OR"}}}}'` ，表示待搜索的文档中包含"redis"或"cluster"即满足查询条件。
	  ▪ fuzziness：是否开启模糊匹配查询，默认为关闭， `"fuzziness":1` 为开启模糊匹配，例如 `'{"query":{"match":{"f0":{"query":"excelentl","fuzziness":1}}}}'` 。
	  ▪ prefix_length：设置模糊匹配保持不变的起始字符数，默认为0，该参数仅在开启模糊匹配时生效。
	  ▪ term：词根查询，不会对查询内容进行拆分，效率比 `match` 高，支持查询所有字段类型。例如 `'{"query":{"term":{"f0":{"value":"redis","boost": 2.0}}}}'` 或 `'{"query":{"term":{"f0":"redis"}}}'` 。
	  ▪ terms：多词根查询，例如 `'{"query":{"terms":{"f0":["redis","excellent"]}}}'` 。
	  ▪ range：范围查询，支持所有字段类型，取值为 `lt` （小于）、 `gt` （大于）、 `lte` （小于等于）和 `gte` （大于等于），例如 `'{"query":{"range":{"f0":{"lt":"abcd","gt":"ab"}}}}'` 。
	  ▪ wildcard：通配符查询，格式为 `"wildcard":{"待查询field":{"value":"查询内容"}}` ，只支持 `*` 和 `?` 通配符，例如 `'{"query":{"wildcard":{"f0":{"value":"*b?Se"}}}}'` 。
	  ▪ prefix：前缀查询，例如 `'{"query":{"prefix":{"f0":"abcd"}}}'` ，表示查询f0字段以 `abcd` 为开头的文档。

类别	说明
选项	■ bool：复合查询，取值如下。 ■ must：查询结果需匹配must列表中所有查询子句。 ■ must_not：查询结果需不匹配must_not列表中所有查询子句。 ■ should：查询结果需匹配should列表中任意一个查询子句（表示或），可指定minimum_should_match参数，表示至少需匹配多少个查询子句。 例如 `'{"query":{"bool":{"must":[{"match":{"DB":"redis"}},{"match":{"Architectures":"cluster"}}]}}}'` ，表示查询DB为Redis、架构（Architectures）为cluster的文档。 ■ dis_max：最佳匹配复合查询，默认返回queries查询子句中单个query分数最高的结果在首位。若指定了tie_breaker参数，将增加匹配多个查询子句的文档分数，tie_breaker的取值范围为[0，1]，默认为0。指定tie_breaker后，目标文档分数的计算规则为：query的最高分+（（将除最高分以外的query分数相加）* tie_breaker）。例如查询中有3个子句，tie_breaker为0.5，doc1在每个子句的分数为[5,1,3]，最终doc1分数为5+((1+3)*0.5)。 命令示例为 `'{"query":{"dis_max":{"queries":[{"term":{"f0":"redis"}},{"term":{"f1":"database"}},{"match":{"title":"redis memory database"}}],"tie_breaker": 0.5}}}'` 。 ■ constant_score：固定分数的复合查询，通过filter查询并返回任意一个符合的文档，支持设置权重（boost，double类型，默认为1.0），文档将以指定的boost返回。例如 `'{"query":{"constant_score":{"filter":{"term":{"f0":"abc"}},"boost":1.2}}}'` 。 ■ match_all：查询全部文档，支持设置权重（boost，double类型，默认为1.0），可以将指定的分数赋予所有返回的文档，例如 `'{"query":{"match_all":{"boost":2.3}}}'` 。 **说明** 通过bool、dis_max与constant_score进行复合查询时，可对term、terms、range、wildcard设置权重（boost，默认为1），区分多个条件的权重，例如 `'{"query":{"bool":{"must":[{"match":{"f0":"v0"}},"term":{"f0":{"value":"redis","boost":2}}]}}}'` 。 ○ sort：结果排序，取值如下。 ■ _score（默认）：按计分权重降序，示例为 `"sort":"_score"` 。 ■ _doc：按文档ID升序，示例为 `"sort":"_doc"` 。 ■ 指定字段：按指定字段升序，该字段必须为索引（index为true）字段或开启强制类型检查（enabled为true）的字段，同时仅支持指定字段类型为：keyword、long、integer、double。例如 `"sort":"price"` ，表示指定以price字段升序排序。 支持输入一个数组进行多条件排序，将按照数组内的顺序进行排序，可通过order调整默认顺序，取值为desc（降序）和acs（升序）。例如 `'{"sort":[{"price":{"order":"desc"}},{"_doc":{"order":"desc"}}]}'` ，表示优先根据price排序，在price相同时根据_doc排序。 ○ _source：指定查询结果中的字段，可指定includes（需要展示的field）和excludes（不需要展示的field），支持配置通配符模式（WildCard）的field。例如 `"_source":{"includes":["f0"]` ，表示查询结果仅返回f0字段。 ○ track_total_hits：通过term、terms、match（不开启模糊匹配）语句查询（含复合查询）时，是否查询所有文档。取值如下： ■ false（默认）：仅查询score排名前100的文档。 ■ true：查询所有文档。 **警告** 开启后，若命中的文档数过多会导致慢查询，请谨慎使用。 ○ from：指定开始返回的目标文档起点，默认为0，表示从第一个查到的文档开始返回。 ○ size：返回查询的文档数量，默认为10，取值范围为[0,10000]。 **说明** 可通过from和size参数到达分页的效果，但分页越多越影响性能。

类别	说明
返回值	• 执行成功：返回查询到的文档信息。 **说明** 返回参数中的relation为查询到的文档数量，取值：eq（表示查询到的文档数量等于value值）、gte（表示大于或等于value值），示例如下。 `{ "hits": { "hits": [], "max_score": null, "total": { "relation": "gte", "value": 100 } } }` • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.SEARCH idx:product '{"sort":[{"price":{"order":"desc"}}]}'`
	返回示例：
	`'{"hits":{"hits":[{"_id":"fruits_3","_index":"idx:product","_score":1.0,"_source":{"product_id":"fruits_3","product_name":"orange","price":30.2,"stock":3000}},{"_id":"fruits_2","_index":"idx:product","_score":1.0,"_source":{"product_id":"fruits_2","product_name":"banana","price":24.0,"stock":2000}},{"_id":"fruits_1","_index":"idx:product","_score":1.0,"_source":{"product_id":"fruits_1","product_name":"apple","price":19.5,"stock":1000}}],"max_score":1.0,"total":{"relation":"eq","value":3}}}'`

TFT.ADDSUG

类别	说明
语法	`TFT.ADDSUG index text weight [text weight] ...`
命令描述	在指定索引中，添加自动补全的文本及对应权重，支持添加多个文本。
选项	• index：待操作的索引名称。 • text：自动补全文本。 • weight：对应文本的计分权重，范围为正整数。
返回值	• 执行成功：返回成功添加的文档数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.ADDSUG idx:redis 'redis is a memory database' 3 'redis cluster' 10`
	返回示例：
	`(integer) 2`

TFT.DELSUG

类别	说明
语法	`TFT.DELSUG index text [text] ...`
命令描述	在指定索引中，删除自动补全的文本，支持删除多个文本。
选项	• index：待查询的索引名称。 • text：待删除的文本，需指定完整、正确的文本。
返回值	• 执行成功：返回成功删除的文本数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.DELSUG idx:redis 'redis is a memory database' 'redis cluster'`
	返回示例：
	`(integer) 2`

TFT.SUGNUM

类别	说明
语法	`TFT.SUGNUM index`
命令描述	获取指定索引中自动补全文本的数量。
选项	• index：待查询的索引名称。
返回值	• 执行成功：返回索引中的文本数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.SUGNUM idx:redis`
	返回示例：
	`(integer) 3`

TFT.GETSUG

类别	说明
语法	`TFT.GETSUG index prefix [MAX_COUNT count] [FUZZY]`
命令描述	根据指定前缀，查询匹配的自动补全文本，将优先返回权重比较高的text。
选项	• index：待查询的索引名称。 • prefix：指定的查询前缀。 • MAX_COUNT count：配置返回文本的最大数量，count的取值范围为[0,255]。 • FUZZY：是否启用模糊匹配。

类别	说明
返回值	• 执行成功：返回索引中的文本数量。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.GETSUG idx:redis res MAX_COUNT 2 FUZZY`
	返回示例：
	`1) "redis cluster"`
	`2) "redis lock"`

TFT.GETALLSUGS

类别	说明
语法	`TFT.GETALLSUGS index`
命令描述	获取指定索引的全量自动补全文本。
选项	• index：待查询的索引名称。
返回值	• 执行成功：返回自动补全文本的列表。 • 其他情况返回相应的异常信息。

示例	命令示例：
	`TFT.GETALLSUGS idx:redis`
	返回示例：
	`1) "redis cluster"`
	`2) "redis lock"`
	`3) "redis is a memory database"`

3.7. Tair集群无感扩缩容介绍

本文列举社区版Redis集群、云数据库Redis集群版扩缩容方案的不足，并介绍Tair集群版无感扩缩容方案。

社区版Redis集群版通常会涉及到数据节点弹性扩缩容、分片间的数据迁移等需求，但业界常见的扩缩容方案仍存在一些问题，例如按Key迁移速度慢、不支持多Key命令、Lua脚本无法迁移、大Key迁移出现卡顿甚至引发高可用切换、迁移失败回滚复杂等。

Tair推出的新一代以Slot复制为原理的无感迁移数据架构，优化实例内部线程调度算法，高效、准确地控制集群行为，支持真正的无感扩缩容。下文详细介绍了社区版Redis集群、云数据库Redis集群扩缩容方案的不足以及Tair集群版无感扩缩容方案。

社区版Redis集群常见扩缩容方案与不足

- **社区版Redis集群弹性扩缩容**

  社区版Redis集群通过Gossip协议传递数据，以槽（Slot）为数据的最小集合，并以遍历（并迁移）Key的方式迁移单个Slot。但该方案存在如下问题：

  - 稳定性较差
    - 数据按Key迁移，可能会导致同一Slot内涉及多Key的命令在迁移过程中执行失败。
    - 无法同时拷贝Lua脚本，导致迁移后Lua脚本丢失。
    - 数据拷贝方式可能会导致大Key迁移出现卡顿，甚至错误，触发高可用（HA）故障切换。

- 运维难度大
  - 若迁移期间出现失败，需要依靠人工手动恢复数据库，难度高、耗时长且易出错。
  - 按Key迁移通常速度慢，扩缩容所需耗时长，往往迫使业务方在业务低峰期进行扩缩容，甚至影响到业务正常进行。

- **基于数据同步迁移组件进行扩缩容**
  该方案不依靠社区版Redis集群特性进行迁移，而是引入一个中间组件作为数据迁移工具。例如先构建一个新的集群实例，然后通过中间组件进行数据迁移，完成迁移后通过LoadBalancer组件切换访问路径，最终完成扩缩容操作。该方案存在如下问题：
  - 全量数据同步耗时过长。
  - 扩缩容期间需要同时拥有2套集群资源，成本较高。
  - LoadBalancer组件在切换过程中会不可避免地引发客户端连接断开的问题，同时切换生效时间较长，可能会导致长达10秒以上的服务不可用。

云数据库Redis集群版平滑扩缩容方案与不足

云数据库Redis集群版提供了平滑扩缩容解决方案，较好地解决了上述社区版Redis集群版扩缩容存在的部分问题，但该方案也存在如下问题：

> **说明** 该方案支持云数据库Redis云盘版。

- 扩缩容变配期间的数据迁移过程有感知，可能存在秒级别的只读时间。

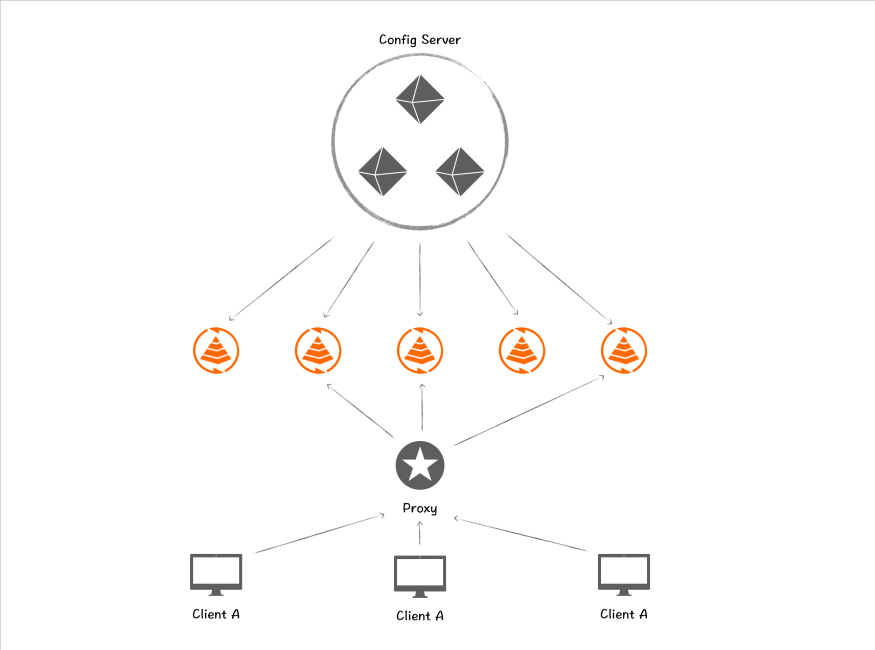
  > **说明** 扩缩容涉及到数据迁移，通常过程为：服务端处理增量同步数据至差值小于一个阈值时，进入只读状态，等待剩余增量数据迁移完成，然后重定向客户端请求到新的地址。若服务端在只读状态期间收到更新请求，则会拒绝请求并返回只读错误，这类错误响应无法被业务逻辑平滑处理，从而影响到业务体验。

- 扩缩容变配的数据迁移任务与正常业务争抢处理资源，迫使用户需要在业务低峰期进行扩缩容操作，影响扩缩容的灵活性。

Tair集群版无感扩缩容方案

Tair（Redis企业版）集群版基于云数据库Redis版新一代管控架构，通过中心化控制组件，高效、准确地控制集群行为，支持真正的无感扩缩容。

> **说明** 本方案支持Tair（Redis企业版）云盘版性能增强型、持久内存型实例。



本方案具有如下特点：

- 真正的无感扩缩容
  Tair集群版在扩缩容过程中可实现客户端无感知、不闪断、无只读状态，满足随时弹性资源伸缩需求。

平滑处理扩缩容的关键在于缩小数据迁移时服务端的只读区间，Tair集群版实例通过动态评估增量差值，将数据库内核的只读区间（时间）控制到毫秒级别，远小于百毫秒级别的TCP重传时间，因此服务端理论上可规避只读错误。服务端只读区间（禁写）时，涉及待迁移Key的写请求将缓存至服务端，等待数据迁移结束后，向客户端返回重定向信息。同时，集群管理系统与数据库内核联动，数据迁移完成后将第一时间更新集群信息。因此，可以实现在客户端完全无感知的情况下完成扩缩容操作。

- 扩缩容过程平滑
  Tair集群版通过优化实例内部的线程调度算法，对迁移任务进行细粒度控制，从而最大化提升线程执行效率，线程执行效率可从原先的10%提升到80%（可配置），实现在不影响业务服务的情况下最大化提升数据迁移速度。同时，Tair集群版在迁移过程中可保障数据高可靠，支持按细粒度扩缩容，且无明显RT（Reaction Time）上涨，杜绝由RT抖动错误触发高可用切换的隐患。
- 运维高效简便
  Tair集群版解决多项社区版Redis集群扩缩容的问题。
  - 后台预拷贝：Tair集群版采用后台预拷贝方式，数据拷贝过程中不会影响线上业务，拷贝完成前源端持有完整数据，规避大Key迁移卡顿等问题。
  - 一键回滚：在扩缩容过程中若发生异常情况，支持一键回滚。
  - 数据按槽（Slot）迁移：数据按Slot迁移，规避同一槽（Slot）内涉及多Key的命令在迁移过程中执行失败的问题。
  - 支持同时拷贝Lua脚本：在迁移过程中会同时拷贝Lua脚本，规避迁移后脚本丢失问题。
  - 支持最大256分片水平扩缩容。
- 低成本
  相比基于中间组件的方案，无需同时持有2套集群资源，具有较低的成本。

4.产品架构

4.1. 架构信息查询导航

云数据库Redis版支持三种架构类型：标准版、集群版与读写分离版。您可根据业务场景选用不同架构的实例。

架构概览

如需了解以下实例架构的详细信息，请单击架构名称跳转到相应的文档。

实例架构	支持的系列	简介	适用场景
标准版-双副本	• 企业版 • 社区版	标准版-双副本采用主从（master-replica）双副本架构，提供高可用切换。	• 对Redis协议兼容性要求较高的业务。 • Redis作为持久化数据存储使用的业务。 • 单个Redis性能压力可控的场景。 • Redis命令相对简单，排序和计算之类的命令较少的场景。
集群版-双副本	• 企业版 • 社区版	双副本集群版实例采用集群架构，每个分片服务器采用主从（master-replica）双副本模式。	• 数据量较大的场景。 • QPS压力较大的场景。 • 吞吐密集型应用场景。
读写分离版	• 企业版 • 社区版	Redis读写分离版本由代理服务器（Proxy Servers）、主备（Master and Replica）节点及只读（Read-Only）节点组成。	• 读取请求QPS压力较大的场景。 • 对Redis协议兼容性要求较高的业务场景。 • Redis作为持久化数据存储使用的业务场景。

文档适用性说明

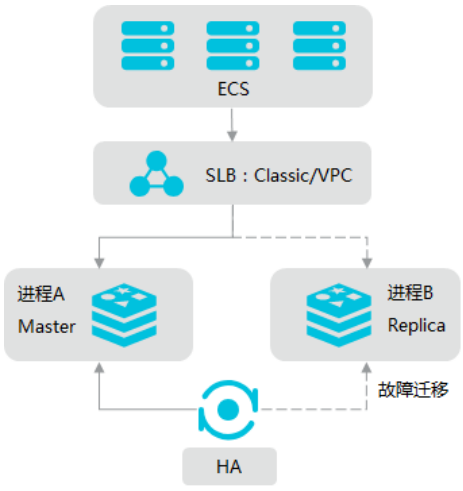
实例架构与产品版本（社区版和企业版）、系列类型（如性能增强型）、引擎版本（Redis版本号，如4.0或5.0）均为不同的概念，架构概览表中涉及的文档适用于各产品版本、系列类型和引擎版本。只要实例的架构是标准版，都可以在标准版-双副本中查看其架构相关介绍，集群版和读写分离版同理。

4.2. 标准版-双副本

标准版-双副本采用主从架构，不仅能提供高性能的缓存服务，还支持数据高可靠。

简介

标准版-双副本模式采用主从（master-replica）模式搭建。主节点提供日常服务访问，备节点提供HA高可用，当主节点发生故障，系统会自动在30秒内切换至备节点，保证业务平稳运行。



特点

- 可靠性

- 服务可靠
  采用双机主从（master-replica）架构，主从节点位于不同物理机。主节点对外提供访问，用户可通过Redis命令行和通用客户端进行数据的增删改查操作。当主节点出现故障，自研的HA系统会自动进行主从切换，保证业务平稳运行。
- 数据可靠
  默认开启数据持久化功能，数据全部落盘。支持数据备份功能，用户可以针对备份集回滚实例或者克隆实例，有效地解决数据误操作等问题。同时，在支持容灾的可用区（例如杭州可用区H+I）创建的实例，还具备同城容灾的能力。

- 兼容性
  标准版完全兼容Redis协议，自建的Redis数据库可以平滑迁移至Redis标准版。阿里云还提供数据传输工具（DTS）支持用户进行增量的Redis迁移，保证业务平稳过渡。
- 阿里云自研
  - 故障探测切换系统（HA）
    阿里云Redis服务封装HA切换系统，实时探测主节点的异常情况，可以有效解决磁盘IO故障，CPU故障等问题导致的服务异常，及时进行主从切换，从而保证服务高可用。
  - 主从复制机制
    阿里云针对Redis主从复制机制进行了定制修改，采用增量日志格式进行复制传输。当主从复制中断后，对系统性能及稳定性影响极低，有效地避免了Redis原生复制的弊端。
    Redis原生复制弊端简要如下：
    - Redis复制中断后，从节点会立即发起psync，psync尝试部分同步，如果不成功，就会全量同步RDB并发送至从节点。
    - 如果Redis全量同步，会导致主节点执行全量备份，进程 Fork，可造成主节点达到毫秒或秒级的卡顿。
    - Redis进程Fork导致Copy-On-Write，Copy-On-Write导致主节点进程内存消耗，极端情况下造成主节点内存溢出，程序异常退出。
    - Redis主节点生成备份文件导致服务器磁盘IO和CPU资源消耗。
    - 发送GB级别大小的备份文件，会导致服务器网络出口爆增，磁盘顺序IO吞吐量高，期间会影响业务正常请求响应时间，并产生其他连锁影响。

使用场景

- 对Redis协议兼容性要求较高的业务
  标准版完全兼容Redis协议，业务可以平滑迁移。
- Redis作为持久化数据存储使用的业务
  标准版提供持久化机制及备份恢复机制，极大地保证数据可靠性。
- 单个Redis性能压力可控的业务
  由于Redis原生采用单线程机制，性能在10万QPS以下的业务建议使用。如果需要更高的性能要求，请选用集群版本。
- Redis命令相对简单，排序、计算类命令较少的业务
  由于Redis的单线程机制，CPU会成为主要瓶颈。如排序、计算类较多的业务建议选用集群版配置。

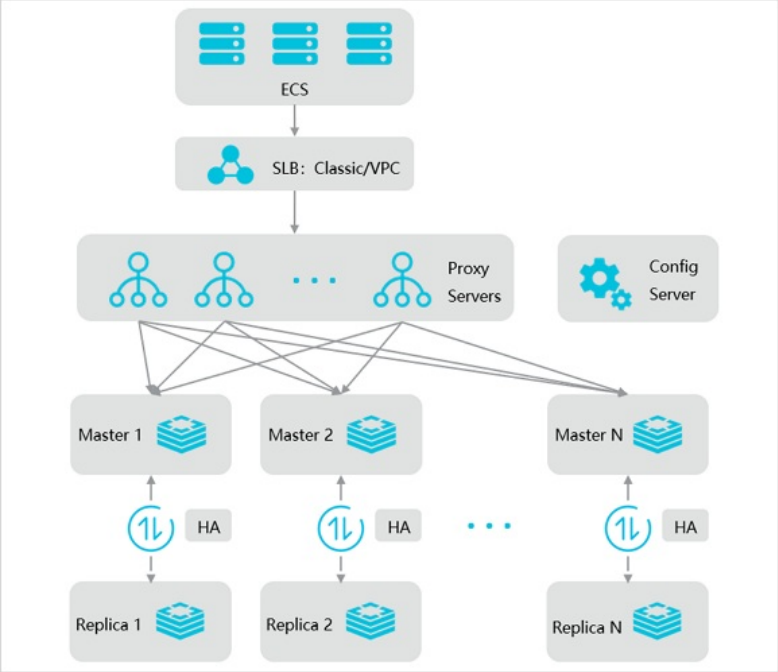
4.3. 集群版-双副本

云数据库Redis版提供双副本集群版实例，可轻松突破Redis自身单线程瓶颈，满足大容量、高性能的业务需求。集群版支持代理和直连两种连接模式，您可以根据本章节的说明，选择适合业务需求的连接模式。

代理模式

集群版默认采用代理（proxy）模式，支持通过一个统一的连接地址（域名）访问Redis集群，客户端的请求通过代理服务器转发到各数据分片，代理服务器、数据分片和配置服务器均不提供单独的连接地址，降低了应用开发难度和代码复杂度。代理模式的服务架构图和组件说明如下。

集群版代理模式服务架构



集群版代理模式组件说明

组件	说明
代理服务器（proxy servers）	单节点配置，集群版结构中会有多个Proxy组成，系统会自动对其实现负载均衡及故障转移。 说明 关于Proxy的详细介绍即特性说明，请参见Redis Proxy特性说明。
数据分片（data shards）	每个数据分片均为双副本（分别部署在不同机器上）高可用架构，主节点发生故障后，系统会自动进行主备切换保证服务高可用。
配置服务器（config server）	采用双副本高可用架构，用于存储集群配置信息及分区策略。

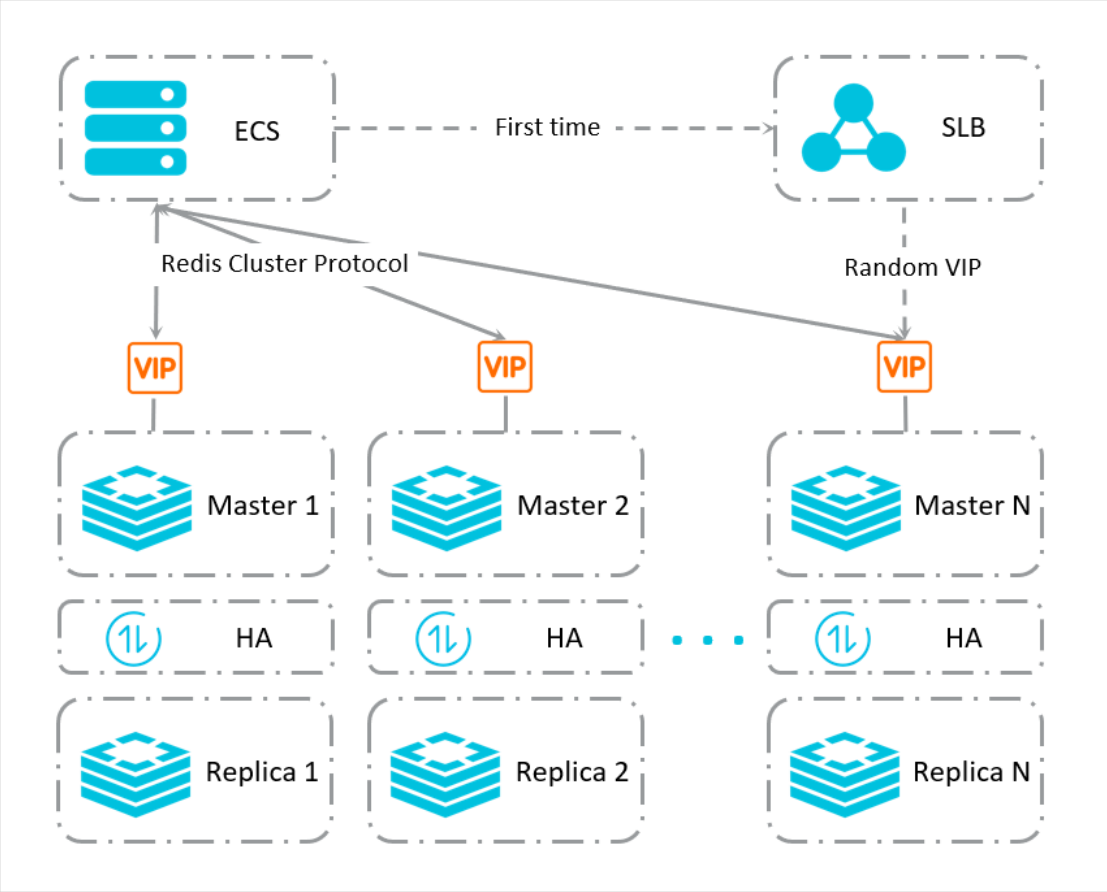
各组件的数量和配置由Redis实例的规格决定，不支持自定义修改，但您可以通过变更实例配置调整集群的大小，或者将实例调整为其它架构。

直连模式

因所有请求都要通过代理服务器转发，代理模式在降低业务开发难度的同时也会小幅度影响Redis服务的响应速度。如果业务对响应速度的要求非常高，您可以使用直连模式，绕过代理服务器直接连接后端数据分片，从而降低网络开销和服务响应时间。直连模式的服务架构和说明如下。

说明 使用直连模式无法享受代理模式带来的好处，例如负载均衡、缓存热点Key信息等特性，更多信息，请参见Redis Proxy特性说明。

集群版直连模式服务架构



使用直连模式需要先开通直连访问，获取直连地址，然后使用连接原生Redis Cluster的方式连接集群。客户端首次连接时会通过DNS将直连地址解析为一个随机数据分片的虚拟IP（VIP）地址，之后即可通过Redis Cluster协议访问各数据分片。直连模式与代理模式的连接方式区别较大，相关注意事项和连接示例请参见使用直连地址访问Redis实例。

使用场景

- 数据量较大
  相比Redis标准版，集群版可以有效地扩展存储量，最大可达4096 GB，能有效的满足业务扩展的需求。
- QPS压力较大
  标准版Redis无法支撑较大的QPS，需要采用多分片的部署方式来突破Redis单线程的性能瓶颈，相关规格请参见集群版-双副本。
- 吞吐密集型应用
  相比Redis标准版，集群版的内网吞吐限制相对较低，可以更好地支持热点数据读取、大吞吐类业务。
- 对Redis协议不敏感的应用
  集群版的架构引入了多个组件，在对Redis协议的支持上相比标准版有一定限制。详情请参见集群架构实例的命令限制。

常用帮助

- 集群版中分片内存占用异常的排查方法请参见如何搜索过大的key。
- 需要掌握内存中数据的分布情况请参见离线全量Key分析。

常见问题

直连模式和代理模式可以同时使用吗？

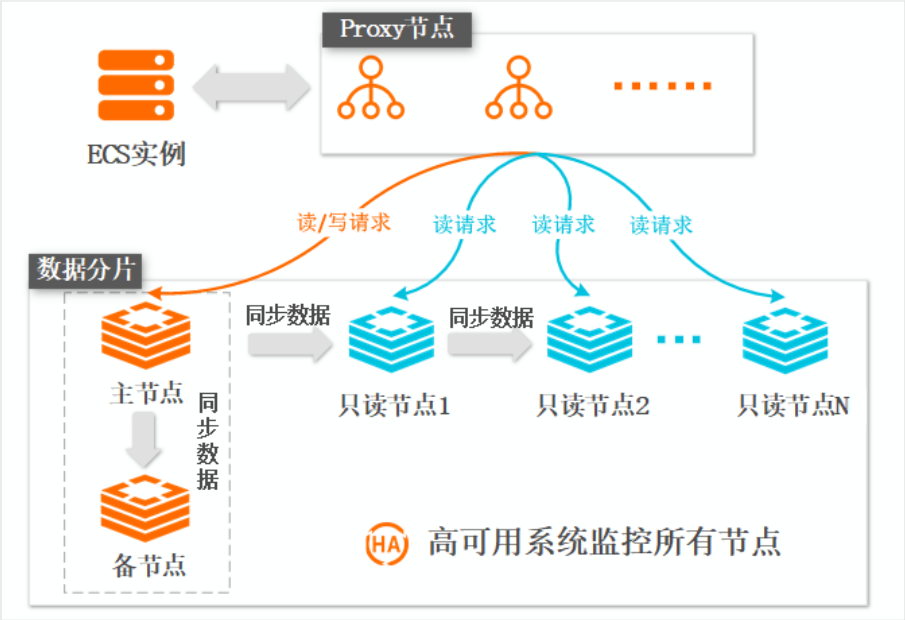
答：本地盘集群架构可以同时使用直连模式和代理模式，云盘集群架构不支持同时使用，只能使用直连模式或代理模式。

4.4. 读写分离版

针对读多写少的业务场景，云数据库Redis推出了读写分离版的产品形态，提供高可用、高性能、灵活的读写分离服务，满足热点数据集中及高并发读取的业务需求，最大化地节约运维成本。

组件介绍

读写分离版主要由主备节点、只读节点、Proxy（代理）节点和高可用系统组成。



组件	说明
主节点	承担写请求的处理，同时和只读节点共同承担读请求的处理。
备节点	作为数据备份使用，不对外提供服务。
只读节点	承担读请求的处理。只读节点采取链式复制架构，扩展只读节点个数可使整体实例性能呈线性增长。同时，采用优化后的binlog执行数据同步，可最大程度地规避全量同步。
Proxy（代理）节点	客户端和Proxy节点建立连接后，Proxy节点会自动识别客户端发起的请求类型，按照权重负载均衡（暂不支持自定义权重），将请求转发到不同的数据节点中。例如将写请求转发给主节点，将读请求转发给主节点或只读节点。 说明 • 客户端需和Proxy节点建立连接，暂不支持和各节点直接建立连接。 • Proxy节点会将读请求平均分配到主节点和只读节点，暂不支持自定义控制。例如，3个只读节点的实例，主节点和3个只读的读权重均为25%。
高可用系统	自动监控各节点的健康状态，异常时发起主备切换或重搭只读节点，并更新相应的路由及权重信息。

特点

- 高可用
  - 通过阿里云自研的高可用系统自动监控所有数据节点的健康状态，为整个实例的可用性保驾护航。主节点不可用时自动选择新的主节点并重新搭建复制拓扑。某个只读节点异常时，高可用系统能够自动探知并重新启动新节点完成数据同步，下线异常节点。
  - Proxy节点实时感知每个只读实例的服务状态。在某个只读实例异常期间，Proxy会自动降低该节点的服务权重，发现只读节点连续失败超过一定次数以后，会停止异常节点的服务权利，并具备继续监控后续重新启动节点服务的能力。
- 高性能
  读写分离版采取链式复制架构，可以通过扩展只读实例个数使整体实例性能呈线性增长，同时基于源码层面对Redis复制流程的定制优化，可以最大程度地提升线性复制的系统稳定性，充分利用每一个只读节点的物理资源。

使用场景

- 读取请求QPS（Queries Per Second）压力较大

标准版Redis无法支撑较大的QPS，如果业务类型是读多写少类型，需要采用多个只读节点的部署方式来突破Redis单线程的性能瓶颈。Redis集群版提供1个、3个、5个只读节点的配置，相比标准版可以将QPS提升近5倍。

> **说明** 由于数据同步至只读节点存在一定延迟，且采用链式复制，只读节点数越多，靠近链路末端的只读节点数据延迟越大，因此选用此架构时，业务需要能接受一定程度的脏数据。如果对数据一致性要求较高，推荐选用集群架构。

- 对Redis协议兼容性要求较高的业务
  读写分离版完全兼容Redis协议命令，可将自建Redis数据库迁移至读写分离版，同时支持从Redis标准版（双副本）一键平滑升级至读写分离版。

> **说明** 读写分离版存在一些命令限制，更多信息，请参见读写分离实例的命令限制。

建议与使用须知

- 当一个只读节点发生故障时，请求会转发到其他节点；如果所有只读节点均不可用，请求会全部转发到主节点。只读节点异常可能导致主节点负载提高、响应时间变长，因此在读负载高的业务场景建议使用多个只读节点。
- 只读节点发生异常时，高可用系统会暂停异常节点的服务，重新挂载一个可用的只读节点。该过程涉及资源分配、实例创建数据同步以及服务加载，消耗的时间与业务负载及数据量有关。云数据库Redis版不承诺只读节点的恢复时间指标。
- 某些场景会触发只读节点的全量同步，例如在主节点触发高可用切换后。全量同步期间只读节点不提供服务并返回 `-LOADING Redis is loading the dataset in memory\r\n` 信息。
- 主节点依然遵从云数据库Redis版的服务等级协议。

相关文档

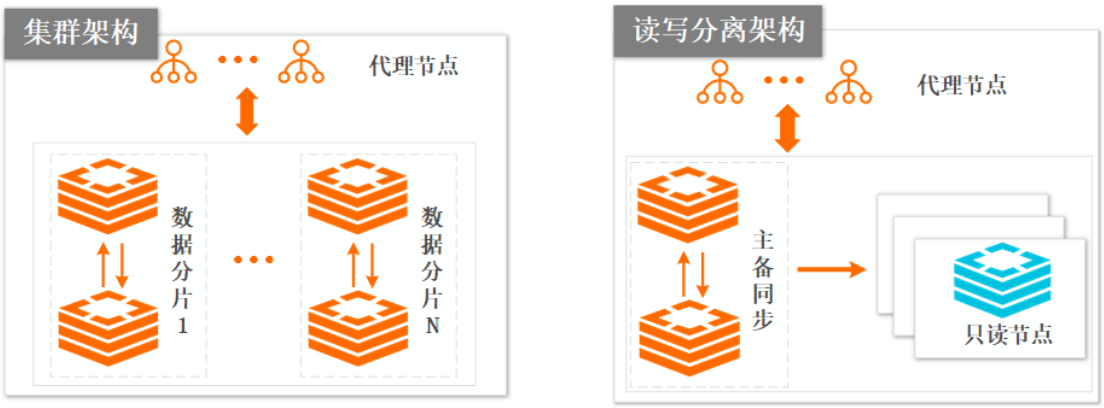
- Redis读写分离技术解析
- 读写分离实例的命令限制

4.5. Redis Proxy特性说明

在云数据库Redis的集群和读写分离架构中，代理服务器（Proxy）承担着路由转发、负载均衡与故障转移等职责。通过了解Proxy的路由转发规则和特定命令的处理方式，有助于您设计更高效的业务系统。

Proxy介绍

集群和读写分离架构图



代理服务器（Proxy）是Redis实例中的一个组件（单节点架构），不会占用数据分片的资源，通过多个Proxy节点实现负载均衡及故障转移。

Proxy能力	说明
集群版使用模式转换	Proxy能够实现架构转换，帮助您如同在使用标准版一样地使用集群版。当标准版无法支撑业务发展时，您无需修改代码即可将标准版的数据迁移至带有Proxy的集群版，大幅度降低业务改造成本。
负载均衡和路由转发	Proxy与后端的数据分片建立长连接，负责请求负载均衡和路由转发操作，关于转发规则的介绍，请参见Proxy的路由转发规则。

Proxy能力	说明
管理只读节点流量	Proxy会实时探测只读节点的状态，当出现下述情况时，Proxy会执行流量管控动作： • 只读节点处于异常状态：Proxy会降低该节点的服务权重，如果多次无法连接该节点，Proxy会停止该节点的服务（即不再将流量转发至该节点），待该异常被修复后重新启用该节点。 • 只读节点处于全量同步状态：Proxy会暂时停止该节点的服务，直到该节点完成全量同步。
缓存热点Key信息	开启代理查询缓存功能（Proxy Query Cache）后，Proxy会缓存热点Key对应的请求和返回信息，当在有效时间内收到同样的请求时直接返回结果至客户端，无需和后端的数据分片交互，可更好地改善对热点Key的发起大量读请求导致的访问倾斜。更多信息，请参见通过Proxy Query Cache优化热点Key问题。 **说明** 该功能仅在集群架构的企业版（性能增强型）实例中支持。
支持多数据库（DB）	集群模式下，原生Redis和Cluster client均不支持多数据库（DB）功能，只使用默认的 `0` 号数据库，也不支持 `SELECT` 命令。但您可以通过Proxy访问集群实例，支持多数据库（DB）功能，支持使用 `SELECT` 命令，集群版实例默认为256个DB。

说明 由于Proxy的演进，Proxy的个数并不完全代表Proxy处理能力，阿里云会保证集群规格中Proxy的配比符合规格说明的要求。

Proxy的路由转发规则

说明 关于各类命令的介绍，请参见Redis命令支持概览。

架构	转发规则	说明
集群架构	基础转发规则	• 对于操作单个Key的命令，Proxy会根据Key所属的Slot（槽）将请求发送给所属的数据分片。 • 对于操作多个Key的命令，如果这些Key是储存在不同的数据分片，Proxy会将命令拆分成多个命令分别发送给对应的分片。
	特定命令转发规则	• 发布订阅类命令 对于**PUBLISH**、**SUBSCRIBE**等发布订阅命令，Proxy会根据channel name进行Hash计算，并分片至对应数据分片。 **说明** 您可以在控制台的性能监控页面选择**数据节点**，然后将自定义监控项设置为**Pub/Sub监控组**，即可查看各数据分片（默认展示第一个数据分片）中发布与订阅（Pub/Sub）相关命令的监控信息。具体操作，请参见自定义监控项（旧版）。 • 阿里云自研的命令 使用阿里云自研的命令（例如**IINFO**、**ISCAN**等）时，如果通过idx参数指定了数据分片ID，Proxy会将这些命令发送到指定的数据分片。更多信息，请参见阿里云自研的Redis命令。
读写分离架构	基础转发规则	• 写请求：Proxy将其直接转发到Master节点。 • 读请求：Proxy将读请求平均分配到主节点和只读节点，暂不支持自定义控制。例如拥有3个只读节点的实例，主节点和3个只读的读权重均为25%。 **说明** SLOWLOG和DBSIZE也属于读命令。

架构	转发规则	说明
读写分离架构	特定命令转发规则	• SCAN类命令 Proxy会将**HSCAN**、**SSCAN**、**ZSCAN**命令发送到到，当所有只读节点均出现异常时，这些命令会被转发至主节点。第一个只读节点 • 阿里云自研的命令 使用阿里云自研的命令（例如**RIINFO**、**RIMONITOR**等）时，可通过ro_slave_idx参数指定该命令要转发到的只读节点，通过idx参数指定该命令要转发到的数据分片。更多信息，请参见阿里云自研的Redis命令。 • 其他命令 Proxy会将事务命令（**MULTI**或**EXEC**）、Lua脚本命令（**EVAL**或**EVALSHA**）、**SCAN**、**INFO**、发布订阅命令（**PUBLISH**、**SUBSCRIBE**等）转发至主节点。

连接数使用说明

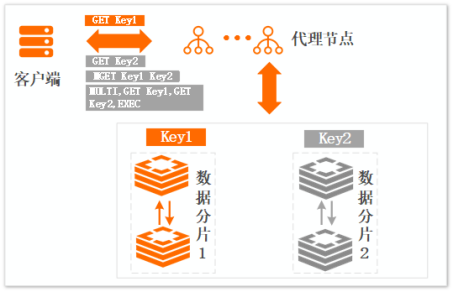
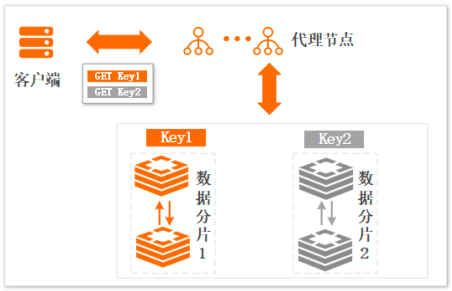
通常情况下，Proxy通过与数据分片建立长连接来处理请求，当请求中包含以下命令时，Proxy会根据命令的处理需求在相应的数据分片上创建额外的连接，您需要合理使用下述命令，避免连接数超限。

> **说明** 代理模式下，社区版实例每个数据分片的连接数上限为10,000，企业版实例每个数据分片的连接数上限为30,000。

- 阻塞类命令：**BRPOP**、**BRPOPLPUSH**、**BLPOP**、**BZPOPMAX**、**BZPOPMIN**。
- 事务类命令：**MULTI**、**EXEC**、**WATCH**。
- MONITOR类命令：**MONITOR**、**IMONITOR**、**RIMONITOR**。
- 订阅命令：**SUBSCRIBE**、**UNSUBSCRIBE**、**PSUBSCRIBE**、**PUNSUBSCRIBE**。
- **SCAN**命令：在多个DB（数据库）中执行SCAN操作。

常见问题

- Q：代理（Proxy）模式下，支持哪些跨Slot的多Key命令？
  A：具体为**DEL**、**EXISTS**、**MGET**、**MSET**、**SDIFF**、**SDIFFSTORE**、**SINTER**、**SINTERSTORE**、**SUNION**、**SUNIONSTORE**、**UNLINK**。
- Q：对于只进行读操作的Lua脚本，支持将流量分摊至只读节点吗？
  A：支持。您需要将实例的readonly_lua_route_ronode_enable的值设置为*1*，即仅包含读操作的Lua脚本会被转发到只读副本处理。具体操作，请参见设置实例参数。
- Q：代理（Proxy）模式和直连模式有什么区别，推荐使用什么模式？
  A：推荐使用代理模式，介绍与区别如下：
  - 集群架构和读写分离架构的实例默认提供代理（Proxy）连接地址，客户端的请求由代理节点转发至数据分片，可享受代理节点带来的负载均衡、读写分离、故障转移、代理查询缓存、长连接等特性能力。
  - 对于集群架构的实例，可申请直连地址，通过该地址可绕过代理，直接访问后端的数据分片（类似连接原生Redis集群）。相比代理模式，直连模式节约了通过代理处理请求的时间，可以在一定程度上提高Redis服务的响应速度。
- Q：如果后端的某个数据分片出现异常，对数据读写有什么影响？
  A：数据分片均采用主备高可用架构，当主节点发生故障后，系统会自动进行主备切换保证服务高可用。在特别极端场景下某个数据分片出现异常后，对数据的影响及优化方案如下。

场景	影响与优化方案
多Key命令场景 	○ 影响： 客户端通过4个连接发送4个请求，当数据分片2处于异常状态时，仅有请求1（**GET Key1**可正常读取到数据），其他请求会访问到数据分片2会返回超时。 ○ 优化方案： ■ 降低多Key命令（例如**MGET** ）的使用频率，或降低一次请求中包含的Key的数量，避免因单个数据分片异常导致该请求全部返回失败。 ■ 降低事务类命令的使用频率或降低事务大小，避免因某个子事务失败导致整个事务失败。
单连接场景 	○ 影响： 客户端通过1个连接分别发送2个请求，当数据分片2处于异常状态时，请求2（**GET Key2**）将返回超时，同时由于请求1（**GET Key1**）和请求2共用同一连接，导致请求1也无法正常返回。 ○ 优化方案： ■ 避免或降低对**pipeline**的使用。 ■ 避免使用单连接的客户端（例如Lettuce），推荐使用连接池的客户端，例如Jedis客户端（需设置合理的超时时间和连接池大小）。

5.实例规格

5.1. 规格查询导航

云数据库Redis版具备多种类型、系列和架构，您可以通过本文的导航信息快速找到相关类型实例的规格文档。

本地盘实例规格

云Redis版的规格文档基于产品版本、系列类型和架构类型进行了分类，您可以单击下表中的架构类型跳转到相应的文档查看详细的规格信息。

类型	系列	实例规格文档	简介
社区版	无	标准版-双副本	主从（master-replica）架构的Redis实例。内存容量上限可达64 GB，支持约80,000 QPS（参考值）。
		集群版-双副本	集群版Redis实例，每个数据分片都是主从（master-replica）架构。最高规格的实例包含32个数据分片，内存容量可达512 GB，支持约2,560,000 QPS（参考值）。
		读写分离版	由一个主从架构的主数据节点、一个或多个只读副本组成的Redis实例。最高规格支持64 GB内存容量、5个只读副本。
		读写分离集群版（已停售）	由一个Redis集群和若干只读副本组成的读写分离实例，集群的每个分片都对应一个只读副本。最高规格支持512 GB内存容量、32个分片、32个只读副本。
企业版（Tair）简介	性能增强型	标准版	采用多线程模型的主从双副本实例，性能约为同规格社区版实例的3倍。内存容量上限可达64 GB，支持约240,000 QPS（参考值）。
		集群版	采用多线程模型的集群实例，每个数据分片均为主从双副本架构，性能约为同规格社区版实例的3倍。内存容量上限可达4096 GB，支持约61,440,000 QPS（参考值）。
		读写分离版	采用多线程模型的读写分离实例，由一个主从架构的主数据节点、一个或多个只读副本组成的Redis实例，性能约为同规社区版实例的3倍。最高规格支持64 GB内存容量、5个只读副本。
早期已停售规格	请参见文档	请参见文档	云数据库Redis版的部分规格已停止新购，但这些规格的已购实例仍可正常使用。您可以在早期已停售规格中查看这些规格的连接数限制、带宽、QPS参考值等信息。

云盘实例规格

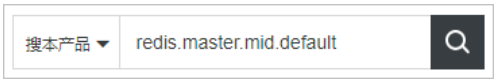
类型	系列	实例规格文档	简介
社区版	无	社区版（云盘）	支持标准架构和集群架构： • 标准架构：内存容量上限可达64 GB，支持约100,000 QPS。 • 集群架构：内存容量上限可达2048 GB，实例整体的性能=分片数x各分片的规格对应的性能。
企业版	性能增强型	性能增强型	支持标准架构和集群架构。采用多线程模型，性能约为同规格社区版实例的3倍，内存容量上限可达64 GB，支持约240,000 QPS。
	持久内存型	持久内存型	支持标准架构和集群架构。数据持久化不依赖传统磁盘，保证每个操作持久化的同时提供近乎Redis社区版的吞吐和延时。标准版的持久内存型实例，单实例成本对比Redis社区版最高可降低30%。
	容量存储型	标准版	主从双副本的容量存储型实例，成本最低为Redis社区版的15%，适用于兼容Redis、需要大容量且较高访问性能的温冷数据存储场景。

常见问题

- 选择规格时，是否需要预留快照内存资源?

答：不需要。目前云数据库Redis社区版与企业版（Tair）均为实例售卖模式，您无需在选型时预留快照所需的内存资源。规格对应的内存容量即为用户最大可用的容量，包含用户数据占用的内存、数据库运行静态内存消耗以及网络链路占用的内存。

- 实例规格都有QPS参考值，如果超过了这个值会有什么影响？
  答：会导致请求堆积。如果长期超过QPS参考值，建议选择更高规格。关于如何变更规格，请参见变更实例配置。
- 为什么找不到某个规格？
  答：您寻找的规格可能已下线。更多信息，请参见早期已停售规格。
- 怎么通过Redis规格代码（InstanceClass）查找规格？
  答：您可以在阿里云文档标题上方的搜索栏中输入规格代码进行查询。



- 怎么测试这些Redis实例的性能？
  答：您可以根据性能白皮书中介绍的方法测试Redis实例的性能。更多信息，请参见性能白皮书。

相关文档

- Redis企业版与社区版特性对比
- 本地盘和云盘实例对比

5.2. 社区版

5.2.1. 标准版-双副本

本章节介绍云数据库Redis社区版标准版-双副本实例的规格，包含内存容量、连接数、带宽、QPS参考值等信息。

实例规格

规格	InstanceClass （API 使用）	CPU核数	每秒新建连接数上限	连接数上限	带宽（MB/s）	QPS参考值
256MB主从版	redis.master.micro.default	2	10,000	10,000	10	80,000
1GB主从版	redis.master.small.default	2	10,000	10,000	10	80,000
2GB主从版	redis.master.mid.default	2	10,000	10,000	16	80,000
4GB主从版	redis.master.stand.default	2	10,000	10,000	24	80,000
8GB主从版	redis.master.large.default	2	10,000	10,000	24	80,000
16GB主从版	redis.master.2xlarge.default	2	10,000	10,000	32	80,000
32GB主从版	redis.master.4xlarge.default	2	10,000	10,000	32	80,000
64GB主从版	redis.master.8xlarge.default	2	10,000	10,000	48	80,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和。
- 读写分离实例的总带宽（即表中显示的带宽）上限为2,048 MB/s。达到该上限后，即使选择拥有更多节点数的实例规格，带宽也不会提高。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB/s，则该规格实例的上下行带宽都是10 MB/s。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

常见问题

- 为什么实例的实际带宽与文档不一致？
  答：部分规格的带宽可能因产品升级发生变化。如果发现实例的带宽与本章节的说明不同，请变更实例配置（可选择与当前相同的规格为变配目标，费用不变）更新带宽。
- 如何创建256MB的实例？
  答：在创建实例页面选择包年包月的计费方式即可找到该规格。

5.2.2. 集群版-双副本

本章节介绍云数据库Redis社区版集群版-双副本实例的规格，包含内存容量、连接数、带宽、QPS参考值等信息。

实例规格

实例规格导航：2分片、4分片、8分片、16分片、32分片、64分片、128分片和256分片。

- **2分片**
  该实例规格的均为2，CPU核数均为4，每秒新建连接数上限均为20,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
2 GB集群版（2分片）	redis.logic.sharding.1g.2db.0rodb.4proxy.default	1	480,000	20,000	96	48	200,000
4 GB集群版（2分片）	redis.logic.sharding.2g.2db.0rodb.4proxy.default	2	480,000	20,000	192	96	200,000
8 GB集群版（2分片）	redis.logic.sharding.4g.2db.0rodb.4proxy.default	4	480,000	20,000	192	96	200,000
16 GB集群版（2分片）	redis.logic.sharding.8g.2db.0rodb.4proxy.default	8	480,000	20,000	192	96	200,000
32 GB集群版（2分片）	redis.logic.sharding.16g.2db.0rodb.4proxy.default	16	480,000	20,000	192	96	200,000

- **4分片**
  该实例规格的均为4，CPU核数均为8，每秒新建连接数上限均为40,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
4 GB集群版（4分片）	redis.logic.sharding.1g.4db.0rodb.4proxy.default	1	480,000	40,000	192	48	400,000

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
8 GB集群版（4分片）	redis.logic.sharding.2g.4db.0rodb.4proxy.default	2	480,000	40,000	384	96	400,000
16 GB集群版（4分片）	redis.logic.sharding.4g.4db.0rodb.4proxy.default	4	480,000	40,000	384	96	400,000
24 GB集群版（4分片）	redis.logic.sharding.6g.4db.0rodb.4proxy.default	6	480,000	40,000	384	96	400,000
32 GB集群版（4分片）	redis.logic.sharding.8g.4db.0rodb.4proxy.default	8	480,000	40,000	384	96	400,000
64 GB集群版（4分片）	redis.logic.sharding.16g.4db.0rodb.4proxy.default	16	480,000	40,000	384	96	400,000
128 GB集群版（4分片）	redis.logic.sharding.32g.4db.0rodb.8proxy.default	32	480,000	40,000	384	96	400,000

- **8分片**

  该实例规格的均为8，CPU核数均为16，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
8 GB集群版（8分片）	redis.logic.sharding.1g.8db.0rodb.8proxy.default	1	500,000	80,000	384	48	800,000
16 GB集群版（8分片）	redis.logic.sharding.2g.8db.0rodb.8proxy.default	2	500,000	80,000	768	96	800,000
32 GB集群版（8分片）	redis.logic.sharding.4g.8db.0rodb.8proxy.default	4	500,000	80,000	768	96	800,000
48 GB集群版（8分片）	redis.logic.sharding.6g.8db.0rodb.8proxy.default	6	500,000	80,000	768	96	800,000
64 GB集群版（8分片）	redis.logic.sharding.8g.8db.0rodb.8proxy.default	8	500,000	80,000	768	96	800,000
128 GB集群版（8分片）	redis.logic.sharding.16g.8db.0rodb.8proxy.default	16	500,000	80,000	768	96	800,000

- **16分片**

  该实例规格的均为16，CPU核数均为32，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
16 GB集群版（16分片）	redis.logic.sharding.1g.16db.0rodb.16proxy.default	1	500,000	160,000	768	48	1,600,000
32 GB集群版（16分片）	redis.logic.sharding.2g.16db.0rodb.16proxy.default	2	500,000	160,000	1,536	96	1,600,000
64 GB集群版（16分片）	redis.logic.sharding.4g.16db.0rodb.16proxy.default	4	500,000	160,000	1,536	96	1,600,000
96 GB集群版（16分片）	redis.logic.sharding.6g.16db.0rodb.16proxy.default	6	500,000	160,000	1,536	96	1,600,000
128 GB集群版（16分片）	redis.logic.sharding.8g.16db.0rodb.16proxy.default	8	500,000	160,000	1,536	96	1,600,000
256 GB集群版（16分片）	redis.logic.sharding.16g.16db.0rodb.16proxy.default	16	500,000	160,000	1,536	96	1,600,000

- **32分片**

  该实例规格的均为32，CPU核数均为64，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
64 GB集群版（32分片）	redis.logic.sharding.2g.32db.0rodb.32proxy.default	2	500,000	320,000	2,048	96	3,200,000
128 GB集群版（32分片）	redis.logic.sharding.4g.32db.0rodb.32proxy.default	4	500,000	320,000	2,048	96	3,200,000
192 GB集群版（32分片）	redis.logic.sharding.6g.32db.0rodb.32proxy.default	6	500,000	320,000	2,048	96	3,200,000
256 GB集群版（32分片）	redis.logic.sharding.8g.32db.0rodb.32proxy.default	8	500,000	320,000	2,048	96	3,200,000
512 GB集群版（32分片）	redis.logic.sharding.16g.32db.0rodb.32proxy.default	16	500,000	320,000	2,048	96	3,200,000

- **64分片**

  该实例规格的均为64，CPU核数均为128，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
128 GB集群版（64分片）	redis.logic.sharding.2g.64db.0rodb.64proxy.default	2	500,000	640,000	2,048	96	6,400,000
256 GB集群版（64分片）	redis.logic.sharding.4g.64db.0rodb.64proxy.default	4	500,000	640,000	2,048	96	6,400,000
512 GB集群版（64分片）	redis.logic.sharding.8g.64db.0rodb.64proxy.default	8	500,000	640,000	2,048	96	6,400,000

- **128分片**

  该实例规格的均为128，CPU核数均为256，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
512 GB集群版（128分片）	redis.logic.sharding.4g.128db.0rodb.128proxy.default	4	500,000	1,280,000	2,048	96	12,800,000

- **256分片**

  该实例规格的均为256，CPU核数均为512，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
1024 GB集群版（256分片）	redis.logic.sharding.4g.256db.0rodb.256proxy.default	4	500,000	2,560,000	2,048	96	25,600,000
2048 GB集群版（256分片）	redis.logic.sharding.8g.256db.0rodb.256proxy.default	8	500,000	2,560,000	2,048	96	25,600,000
4096 GB集群版（256分片）	redis.logic.sharding.16g.256db.0rodb.256proxy.default	16	500,000	2,560,000	2,048	96	25,600,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和，每个分片的带宽限制如下：
  - 社区版：单个分片的内存小于等于1 GB时，每个分片的最大带宽为48 MB/s；单个分片的内存大于1 GB时，每个分片的最大带宽为96 MB/s。
  - 企业版：每个分片的最大带宽为96 MB/s。
- 集群架构的实例使用默认的代理连接地址时，带宽上限为2,048 MB/s。达到该上限后，即使增加分片数量，带宽也不会提高。如需应对业务上的网络超大流量，您可以开通直连访问，具体操作，请参见开通直连访问（仅适用于集群架构）。开启直连访问后：
  - 最大连接数：为单个分片的最大连接数*分片数，社区版单个分片的最大连接数为10,000，企业版单个分片的最大连接数为30,000。

- 整体带宽限制：为单个分片的最大带宽*分片数，例如128分片的集群实例（单个分片的内存大于1 GB，单个分片的最大带宽为96 MB/s），开启直连后整体带宽为12,288 MB/s。

- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB，则该规格实例的上下行带宽都是10 MB。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数规则说明

类别	说明
最大连接数	• ：实例最大连接数的上限为500,000，达到该值后，即使增加分片或节点数，最大连接数也不会提高。代理模式 • ：单分片最大连接数的上限为10,000，实例最大连接数的上限为：分片数*10,000。直连模式
每秒新建连接数	每秒新建连接数上限即每秒内可新增的连接数量。例如实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。 **说明** 当您通过直连地址连接实例时，由于绕过了代理节点，连接性能有一定的下降，单个数据分片的每秒新建连接数上限为2,000。例如实例具备4个数据分片，则整体的每秒新建连接数上限为8,000。

常见问题

- Q：为什么实例的实际带宽与文档不一致？
  A：部分规格的带宽可能因产品升级发生变化。如果发现实例的带宽与本章节的说明不同，请变更实例配置（可选择与当前相同的规格为变配目标，费用不变）更新带宽。
- Q：为什么集群实例的内存未使用完却写入失败？
  A：Redis使用Hash算法将Key均匀地写入至不同的分片中，若集群实例中存在大Key，会导致资源分布倾斜，严重时会导致大Key所在的分片被写满，此时集群的部分写入请求可能发生失败。
  您可以通过**性能监控**功能查看并优化各分片的性能指标，更多信息，请参见如何查看Redis集群子节点内存等相关性能。

5.2.3. 读写分离版

本章节介绍云数据库Redis社区版读写分离实例的规格，包含内存容量、连接数、带宽、QPS参考值等信息。

实例规格

> **说明** 下表中的1节点代表该实例规格仅包含1个数据分片。例如规格为**读写分离1G版（1节点3只读）**表示该实例具备1个数据分片，该数据分片中包含3个只读节点。

规格	InstanceClass（API 使用）	CPU核数	只读节点数	带宽（MB/s）	每秒新建连接数上限	最大连接数	QPS参考值
读写分离1G版（1节点1只读）	redis.logic.splitrw.small.1db.1rodb.4proxy.default	4	1	96	20,000	20,000	200,000
读写分离1G版（1节点3只读）	redis.logic.splitrw.small.1db.3rodb.4proxy.default	8	3	192	40,000	40,000	400,000

规格	InstanceClass（API 使用）	CPU核数	只读节点数	带宽（MB/s）	每秒新建连接数上限	最大连接数	QPS参考值
读写分离1G版（1节点5只读）	redis.logic.splitrw.small.1db.5rodb.6proxy.default	12	5	288	50,000	60,000	600,000
读写分离2G版（1节点1只读）	redis.logic.splitrw.mid.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
读写分离2G版（1节点3只读）	redis.logic.splitrw.mid.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
读写分离2G版（1节点5只读）	redis.logic.splitrw.mid.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
读写分离4G版（1节点1只读）	redis.logic.splitrw.stand.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
读写分离4G版（1节点3只读）	redis.logic.splitrw.stand.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
读写分离4G版（1节点5只读）	redis.logic.splitrw.stand.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
读写分离8G版（1节点1只读）	redis.logic.splitrw.large.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
读写分离8G版（1节点3只读）	redis.logic.splitrw.large.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
读写分离8G版（1节点5只读）	redis.logic.splitrw.large.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
读写分离16G版（1节点1只读）	redis.logic.splitrw.2xlarge.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
读写分离16G版（1节点3只读）	redis.logic.splitrw.2xlarge.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
读写分离16G版（1节点5只读）	redis.logic.splitrw.2xlarge.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000

规格	InstanceClass（API 使用）	CPU核数	只读节点数	带宽（MB/s）	每秒新建连接数上限	最大连接数	QPS参考值
读写分离32G版（1节点1只读）	redis.logic.splitrw.4xlarge.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
读写分离32G版（1节点3只读）	redis.logic.splitrw.4xlarge.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
读写分离32G版（1节点5只读）	redis.logic.splitrw.4xlarge.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
读写分离64G版（1节点1只读）	redis.logic.splitrw.8xlarge.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
读写分离64G版（1节点3只读）	redis.logic.splitrw.8xlarge.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
读写分离64G版（1节点5只读）	redis.logic.splitrw.8xlarge.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和。
- 读写分离实例的总带宽（即表中显示的带宽）上限为2,048 MB/s。达到该上限后，即使选择拥有更多节点数的实例规格，带宽也不会提高。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB/s，则该规格实例的上下行带宽都是10 MB/s。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

5.3. 企业版

5.3.1. 性能增强-标准版

本文介绍云数据库Redis企业版性能增强系列标准版架构的规格，包含内存容量、连接数、带宽、QPS参考值等信息。

实例规格

规格	InstanceClass （API使用）	CPU核数	IO线程数	每秒新建连接数上限	最大连接数	带宽（MB/s）	QPS参考值
1GB主从版性能增强	redis.amber.master.small.multithread	6	4	10,000	30,000	96	240,000
2GB主从版性能增强	redis.amber.master.mid.multithread	6	4	10,000	30,000	96	240,000
4GB主从版性能增强	redis.amber.master.stand.multithread	6	4	10,000	30,000	96	240,000
8GB主从版性能增强	redis.amber.master.large.multithread	6	4	10,000	30,000	96	240,000
16GB主从版性能增强	redis.amber.master.2xlarge.multithread	6	4	10,000	30,000	96	240,000
32GB主从版性能增强	redis.amber.master.4xlarge.multithread	6	4	10,000	30,000	96	240,000
64GB主从版性能增强	redis.amber.master.8xlarge.multithread	6	4	10,000	30,000	96	240,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和。
- 读写分离实例的总带宽（即表中显示的带宽）上限为2,048 MB/s。达到该上限后，即使选择拥有更多节点数的实例规格，带宽也不会提高。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB/s，则该规格实例的上下行带宽都是10 MB/s。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

常见问题

为什么实例的连接数上限与文档描述的不同？

答：部分规格的连接数上限可能因产品升级发生变化。如果发现实例的连接数上限与本文的说明不同，请变更实例配置以更新带宽，变配时选择与当前相同的规格为变配目标即可，操作说明请参见变更实例配置。

5.3.2. 性能增强-集群版

本文介绍云数据库Redis企业版性能增强系列集群版架构的规格，包含内存容量、连接数、带宽、QPS参考值等信息。

实例规格

实例规格导航：2分片、4分片、8分片、16分片、32分片、64分片、128分片和256分片。

- **2分片**

该实例规格的均为2，CPU核数均为12，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass （API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
2 GB集群性能增强版（2分片）	redis.amber.logic.sharding.1g.2db.0rodb.6proxy.multithread	1	480,000	60,000	192	96	480,000
4 GB集群性能增强版（2分片）	redis.amber.logic.sharding.2g.2db.0rodb.6proxy.multithread	2	480,000	60,000	192	96	480,000

- **4分片**

该实例规格的均为4，CPU核数均为24，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass （API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
8 GB集群性能增强版（4分片）	redis.amber.logic.sharding.2g.4db.0rodb.12proxy.multithread	2	500,000	120,000	384	96	960,000

- **8分片**

该实例规格的均为8，CPU核数均为48，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass （API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
16 GB集群性能增强版（8分片）	redis.amber.logic.sharding.2g.8db.0rodb.24proxy.multithread	2	500,000	240,000	768	96	1,920,000
32 GB集群性能增强版（8分片）	redis.amber.logic.sharding.4g.8db.0rodb.24proxy.multithread	4	500,000	240,000	768	96	1,920,000
64 GB集群性能增强版（8分片）	redis.amber.logic.sharding.8g.8db.0rodb.24proxy.multithread	8	500,000	240,000	768	96	1,920,000

- **16分片**

该实例规格的均为16，CPU核数均为96，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass （API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
128 GB集群性能增强版（16分片）	redis.amber.logic.sharding.8g.16db.0rodb.48proxy.multithread	8	500,000	480,000	1,536	96	3,840,000
256 GB集群性能增强版（16分片）	redis.amber.logic.sharding.16g.16db.0rodb.48proxy.multithread	16	500,000	480,000	1,536	96	3,840,000

- **32分片**

该实例规格的均为32，CPU核数均为192，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
512 GB集群性能增强版（32分片）	redis.amber.logic.sharding.16g.32db.0rodb.96proxy.multithread	16	500,000	960,000	2,048	96	7,680,000

- **64分片**

  该实例规格的均为64，CPU核数均为384，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass（API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
1024 GB集群性能增强版（64分片）	redis.amber.logic.sharding.16g.64db.0rodb.192proxy.multithread	16	500,000	1,920,000	2,048	96	15,360,000

- **128分片**

  该实例规格的均为128，CPU核数均为768，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass （API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
2048 GB集群性能增强版（128分片）	redis.amber.logic.sharding.16g.128db.0rodb.384proxy.multithread	16	500,000	3,840,000	2,048	96	30,720,000

- **256分片**

  该实例规格的均为256，CPU核数均为1,536，每秒新建连接数上限均为50,000。分片数

规格	InstanceClass （API 使用）	单分片内存（GB）	最大连接数代理模式	最大连接数直连模式	总带宽（MB/s）	单分片带宽（MB/s）	QPS参考值
4096 GB集群性能增强版（256分片）	redis.amber.logic.sharding.16g.256db.0rodb.768proxy.multithread	16	500,000	7,680,000	2,048	96	61,440,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和，每个分片的带宽限制如下：
  - 社区版：单个分片的内存小于等于1 GB时，每个分片的最大带宽为48 MB/s；单个分片的内存大于1 GB时，每个分片的最大带宽为96 MB/s。
  - 企业版：每个分片的最大带宽为96 MB/s。
- 集群架构的实例使用默认的代理连接地址时，带宽上限为2,048 MB/s。达到该上限后，即使增加分片数量，带宽也不会提高。如需应对业务上的网络超大流量，您可以开通直连访问，具体操作，请参见开通直连访问（仅适用于集群架构）。开启直连访问后：
  - 最大连接数：为单个分片的最大连接数*分片数，社区版单个分片的最大连接数为10,000，企业版单个分片的最大连接数为30,000。
  - 整体带宽限制：为单个分片的最大带宽*分片数，例如128分片的集群实例（单个分片的内存大于1 GB，单个分片的最大带宽为96 MB/s），开启直连后整体带宽为12,288 MB/s。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB，则该规格实例的上下行带宽都是10 MB。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数规则说明

类别	说明
最大连接数	• ：实例最大连接数的上限为500,000，达到该值后，即使增加分片或节点数，最大连接数也不会提高。代理模式 • ：单分片最大连接数的上限为30,000，实例最大连接数的上限为：分片数*30,000。直连模式
每秒新建连接数	每秒新建连接数上限即每秒内可新增的连接数量。例如实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。 **说明** 当您通过直连地址连接实例时，由于绕过了代理节点，连接性能有一定的下降，单个数据分片的每秒新建连接数上限为2,000。例如实例具备4个数据分片，则整体的每秒新建连接数上限为8,000。

常见问题

- Q：为什么实例的实际带宽与文档不一致？
  A：部分规格的带宽可能因产品升级发生变化。如果发现实例的带宽与本章节的说明不同，请变更实例配置（可选择与当前相同的规格为变配目标，费用不变）更新带宽。
- Q：为什么集群实例的内存未使用完却写入失败？
  A：Redis使用Hash算法将Key均匀地写入至不同的分片中，若集群实例中存在大Key，会导致资源分布倾斜，严重时会导致大Key所在的分片被写满，此时集群的部分写入请求可能发生失败。
  您可以通过**性能监控**功能查看并优化各分片的性能指标，更多信息，请参见如何查看Redis集群子节点内存等相关性能。

5.3.3. 性能增强-读写分离版

本章节介绍云数据库Redis企业版读写分离实例的规格，包含只读副本数、内存容量、连接数、带宽、QPS参考值等信息。

实例规格

规格	InstanceClass （API使用）	CPU核数	IO线程数	只读节点数	带宽（MB/s）	每秒新建连接数上限	最大连接数	QPS参考值
1G读写分离性能增强版（1节点1只读）	redis.amber.logic.splitrw.small.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	60,000	480,000
2G读写分离性能增强版（1节点1只读）	redis.amber.logic.splitrw.mid.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	60,000	480,000
4G读写分离性能增强版（1节点1只读）	redis.amber.logic.splitrw.stand.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	60,000	480,000
8G读写分离性能增强版（1节点1只读）	redis.amber.logic.splitrw.large.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	60,000	480,000

规格	InstanceClass （API使用）	CPU核数	IO线程数	只读节点数	带宽（MB/s）	每秒新建连接数上限	最大连接数	QPS参考值
16G读写分离性能增强版（1节点1只读）	redis.amber.logic.splitrw.2xlarge.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	60,000	480,000
32G读写分离性能增强版（1节点1只读）	redis.amber.logic.splitrw.4xlarge.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	60,000	480,000
64G读写分离性能增强版（1节点1只读）	redis.amber.logic.splitrw.8xlarge.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	60,000	480,000
1G读写分离性能增强版（1节点3只读）	redis.amber.logic.splitrw.small.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	120,000	960,000
2G读写分离性能增强版（1节点3只读）	redis.amber.logic.splitrw.mid.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	120,000	960,000
4G读写分离性能增强版（1节点3只读）	redis.amber.logic.splitrw.stand.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	120,000	960,000
8G读写分离性能增强版（1节点3只读）	redis.amber.logic.splitrw.large.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	120,000	960,000
16G读写分离性能增强版（1节点3只读）	redis.amber.logic.splitrw.2xlarge.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	120,000	960,000
32G读写分离性能增强版（1节点3只读）	redis.amber.logic.splitrw.4xlarge.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	120,000	960,000
64G读写分离性能增强版（1节点3只读）	redis.amber.logic.splitrw.8xlarge.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	120,000	960,000
1G读写分离性能增强版（1节点5只读）	redis.amber.logic.splitrw.small.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
2G读写分离性能增强版（1节点5只读）	redis.amber.logic.splitrw.mid.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000

规格	InstanceClass （API使用）	CPU核数	IO线程数	只读节点数	带宽（MB/s）	每秒新建连接数上限	最大连接数	QPS参考值
4G读写分离性能增强版（1节点5只读）	redis.amber.logic.splitrw.stand.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
8G读写分离性能增强版（1节点5只读）	redis.amber.logic.splitrw.large.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
16G读写分离性能增强版（1节点5只读）	redis.amber.logic.splitrw.2xlarge.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
32G读写分离性能增强版（1节点5只读）	redis.amber.logic.splitrw.4xlarge.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
64G读写分离性能增强版（1节点5只读）	redis.amber.logic.splitrw.8xlarge.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和。
- 读写分离实例的总带宽（即表中显示的带宽）上限为2,048 MB/s。达到该上限后，即使选择拥有更多节点数的实例规格，带宽也不会提高。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB/s，则该规格实例的上下行带宽都是10 MB/s。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

5.4. 社区版（云盘）

本章节介绍云数据库Redis云盘社区标准版规格，包含内存容量、连接数、带宽、QPS参考值等信息。

性能说明

本文中的规格可适用于标准架构和集群架构的云盘实例。

架构	性能说明
标准架构	实例整体的性能与实例规格表中对应的性能一致。

架构	性能说明
集群架构	实例整体的性能=分片数x各分片的规格对应的性能。例如实例具备4个分片，分片的规格均为redis.shard.small.ce（即实例规格中的第二个规格代码），每个分片的性能为： • CPU核数：2核。 • 带宽：24 MB/s。 • 最大连接数：10,000。 • QPS参考值：100,000。 那么，该实例的整体性能即为： • CPU核数：8核。 • 带宽：96 MB/s。 • 最大连接数：40,000。 • QPS参考值：400,000。

实例规格

> 说明
> - 规格中的ESSD云盘仅用作系统运行使用（例如日志、数据备份等），不作为数据存取的介质。
> - 集群架构的云盘实例支持自由调整分片节点的数量（最少2个，最多256个且规格相同，每次变配分片数量的上限为64个），具体操作，请参见调整云盘实例的分片数量。

实例规格

规格名称	规格代码（API使用）	CPU核数	内存（GB）	ESSD云盘（GB）	带宽（MB/s）	最大连接数	QPS参考值
256 MB （vCPU shared）	redis.shard.micro.ce	2	0.25	1.25	24	10,000	100,000
1 GB （vCPU shared）	redis.shard.small.ce	2	1	5	24	10,000	100,000
2 GB （vCPU shared）	redis.shard.mid.ce	2	2	10	24	10,000	100,000
4 GB （vCPU shared）	redis.shard.large.ce	2	4	20	32	10,000	100,000
8 GB （vCPU shared）	redis.shard.xlarge.ce	2	8	40	40	10,000	100,000
16 GB （vCPU shared）	redis.shard.2xlarge.ce	2	16	80	80	10,000	100,000
24 GB （vCPU shared）	redis.shard.3xlarge.ce	2	24	120	96	10,000	100,000
32 GB （vCPU shared）	redis.shard.4xlarge.ce	2	32	160	96	10,000	100,000
64 GB （vCPU shared）	redis.shard.8xlarge.ce	2	64	320	96	10,000	100,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽说明

- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB，则该规格实例的上下行带宽都是10 MB。
- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制。建议通过专有网络连接Redis实例，可获得更高的安全性和更低的网络延迟。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

常见问题

为什么实例的实际带宽与文档不一致？

答：部分规格的带宽可能因产品升级发生变化。如果发现实例的带宽与本章节的说明不同，请变更实例配置以更新带宽，变配时选择与当前相同的规格为变配目标即可，操作说明请参见变更实例配置。

5.5. 企业版（云盘）

5.5.1. 性能增强型

本文介绍云数据库Redis企业版性能增强型（云盘版）规格，包含内存容量、连接数、带宽、QPS参考值等信息。

> **说明** 关于本地盘和云盘版的详细信息，请参见本地盘和云盘实例对比。

性能说明

本文中的规格可适用于标准架构和集群架构的性能增强型实例。

架构	性能说明
标准架构	实例整体的性能与实例规格表中对应的性能一致。
集群架构	实例整体的性能=分片数×各分片的规格对应的性能。 **说明** 集群架构实例使用代理模式访问时，带宽上限为2,048 MB/s，总QPS上限为10,000,000，最大连接数上限为500,000。如需应对业务上的网络超大流量，您可以使用直连模式访问，具体操作请参见开通直连访问。 例如实例具备4个分片，分片的规格均为tair.rdb.256m（即实例规格表中的第一个规格代码），每个分片的性能为： • CPU核数：6核。 • 带宽：96 MB/s。 • 连接数：30,000。 那么，该实例的整体性能即为： • CPU核数：24核。 • 带宽：384 MB/s。 • 最大连接数：120,000。

实例规格

> **说明** 规格中的ESSD云盘仅用作系统运行使用（例如日志、数据备份等），不作为数据存取的介质。

实例规格

规格名称	规格代码（API使用）	CPU核数	内存（GB）	ESSD云盘（GB）	I/O线程数	带宽（MB/s）	每秒新建连接数	最大连接数	QPS参考值
256 M 增强版	tair.rdb.256m	6	0.25	1.25	4	96	30,000	30,000	300,000
1 GB 增强版	tair.rdb.1g	6	1	5	4	96	30,000	30,000	300,000
2GB 增强版	tair.rdb.2g	6	2	10	4	96	30,000	30,000	300,000
4GB 增强版	tair.rdb.4g	6	4	20	4	96	30,000	30,000	300,000

规格名称	规格代码（API使用）	CPU核数	内存（GB）	ESSD云盘（GB）	I/O线程数	带宽（MB/s）	每秒新建连接数	最大连接数	QPS参考值
8GB 增强版	tair.rdb.8g	6	8	40	4	96	30,000	30,000	300,000
16GB 增强版	tair.rdb.16g	6	16	80	4	96	30,000	30,000	300,000
24GB 增强版	tair.rdb.24g	6	24	120	4	96	30,000	30,000	300,000
32GB 增强版	tair.rdb.32g	6	32	160	4	96	30,000	30,000	300,000
64GB 增强版	tair.rdb.64g	6	64	320	4	96	30,000	30,000	300,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽说明

- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB，则该规格实例的上下行带宽都是10 MB。
- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制。建议通过专有网络连接Redis实例，可获得更高的安全性和更低的网络延迟。

连接数说明

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

相关文档

- 企业版（性能增强型）介绍
- 标准架构介绍

5.5.2. 持久内存型

本文介绍云数据库Redis企业版持久内存型的规格（含标准架构和集群架构），包含内存容量、连接数、带宽等信息。

购买方式

创建Redis企业版云盘实例持久内存型或容量存储型实例

术语介绍

持久内存型

Redis企业版持久内存型（简称持久内存型）基于Intel 傲腾™持久内存，为您提供大容量、兼容Redis的内存数据库产品。单实例成本对比Redis社区版最多可降低30%，且数据持久化不依赖传统磁盘，保证每个操作持久化的同时提供近乎Redis社区版的吞吐和延时，极大提升业务数据可靠性。更多详情，请参见持久内存型。

ESSD云盘

阿里云ESSD（Enhanced SSD）云盘，相关概念及ESSD性能级别请参见ESSD云盘。

> **说明** 规格中的ESSD云盘仅用作系统运行使用（例如日志、数据备份等），不作为数据存取的介质。

实例规格

下表为单个分片的规格，集群实例整体的性能 = 分片数 * 各分片规格对应的性能。

规格	InstanceClass （API 使用）	规格信息		最大连接数	最大带宽（MB/s）	QPS参考值
		CPU核数	持久内存（GB）			
主从4G持久内存	tair.scm.standard.1m.4d	3	4	10,000	96	100,000
主从8G持久内存	tair.scm.standard.2m.8d	3	8	10,000	96	100,000
主从16G持久内存	tair.scm.standard.4m.16d	3	16	10,000	96	100,000
主从32G持久内存	tair.scm.standard.8m.32d	3	32	10,000	96	100,000
主从64G持久内存	tair.scm.standard.16m.64d	3	64	10,000	96	100,000

性能说明

本文中的规格可适用于标准架构和集群架构的持久内存型实例。

架构	性能说明
标准架构	实例整体的性能与实例规格表中对应的性能一致。
集群架构	实例整体的性能 = 分片数 * 各分片的规格对应的性能。 **说明** 集群架构实例使用代理模式访问时，带宽上限为2,048 MB/s，总QPS上限为10,000,000，最大连接数上限为500,000。达到该上限后，即使增加分片数量，性能也不会提高。如需应对业务上的网络超大流量，您可以使用直连模式访问，具体操作请参见开通直连访问。 例如实例具备4个分片，分片的规格均为tair.scm.standard.2m.8d（即实例规格中的第二个规格代码），每个分片的性能为： • CPU核数：3核。 • 带宽：96 MB/s。 • 连接数：30,000。 那么，该实例的整体性能即为： • CPU核数：12核。 • 带宽：384 MB/s。 • 最大连接数：120,000。

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

相关文档

- 持久内存型性能测试
- Tair命令限制

5.5.3. 容量存储型

本文介绍云数据库Redis企业版容量存储型（标准版）的规格，包含内存容量、连接数、带宽等信息。

购买方式

创建Redis企业版云盘实例持久内存型或容量存储型实例

术语介绍

容量存储型

Redis企业版容量存储型（简称容量存储型）基于云盘ESSD研发，兼容Redis核心数据结构与接口，可提供大容量、低成本、强持久化的数据库服务。容量存储型在提升了成本和数据可靠性的同时，也解决了原生Redis固有的fork问题而预留部分内存的问题。适用于兼容Redis、需要大容量且较高访问性能的温冷数据存储场景。更多详情，请参见容量存储型。

ESSD云盘

阿里云ESSD（Enhanced SSD）云盘，相关概念及ESSD性能级别请参见ESSD云盘。

ECS

云服务器ECS（Elastic Compute Service）是阿里云提供的性能卓越、稳定可靠、弹性扩展的IaaS（Infrastructure as a Service）级别云计算服务。Redis容量存储型实例采用的是ECS的通用平衡增强型实例规格族g6e，该规格组依托第三代神龙架构，将大量虚拟化功能卸载到专用硬件，降低虚拟化开销，提供稳定可预期的超高性能。同时通过芯片快速路径加速手段，完成存储、网络性能以及计算稳定性的数量级提升。

相关ECS规格详情，请参见通用平衡增强型实例规格族g6e。

实例规格

InstanceClass（API使用）	规格信息			系统盘		对应的ECS实例规格	最大连接数	最大带宽（GB）
	CPU核数	内存（GB）	存储容量可选范围（GB）	ESSD云盘（GB）	ESSD性能级别			
tair.essd.standard.xlarge	4	16	60~130	40	PL1	ecs.g6e.xlarge	10,000	突发最高0.625
tair.essd.standard.2xlarge	8	32	60~260	40	PL1	ecs.g6e.2xlarge	10,000	突发最高0.625
tair.essd.standard.4xlarge	16	64	60~530	40	PL1	ecs.g6e.4xlarge	30,000	突发最高0.625
tair.essd.standard.8xlarge	32	128	60~1030	40	PL1	ecs.g6e.8xlarge	30,000	突发最高0.625
tair.essd.standard.13xlarge	52	192	60~1540	40	PL1	ecs.g6e.13xlarge	30,000	突发最高0.625

说明

- 目前仅支持标准架构。
- 企业版Tair容量存储型实例的底层操作系统、管理服务均会占用部分内存，因此实例实际可用内存不会达到规格显示的内存大小，根据实际使用情况，可用内存通常为规格显示内存的75% ~ 90%。

相关文档

- 容量存储型性能测试
- Tair命令限制

5.6. 历史规格

5.6.1. 读写分离集群版（已停售）

本章节介绍云数据库Redis社区版读写分离集群实例的规格，包含内存容量、连接数、带宽、QPS参考值等信息。

实例规格

说明 下表中的2节点代表该实例规格包含2个数据分片。例如规格为**读写分离4G版（2节点1只读）**表示该实例具备2个数据分片，每个数据分片中包含1个只读节点。

规格	InstanceClass （API使用）	CPU核数	分片数	每分片只读副本数	带宽（MB/s）	每秒新建连接数上限	最大连接数	QPS参考值
读写分离4G版（2节点1只读）	redis.logic.splitrw.sharding2g.2db.1rodb.4proxy.default	8	2	1	384	40,000	40,000	400,000
读写分离8G版（2节点1只读）	redis.logic.splitrw.sharding4g.2db.1rodb.4proxy.default	8	2	1	384	40,000	40,000	400,000
读写分离16G版（2节点1只读）	redis.logic.splitrw.sharding8g.2db.1rodb.8proxy.default	8	2	1	384	40,000	40,000	400,000
读写分离32G版（8节点1只读）	redis.logic.splitrw.sharding4g.8db.1rodb.16proxy.default	32	8	1	1,536	50,000	160,000	1,600,000
读写分离64G版（16节点1只读）	redis.logic.splitrw.sharding4g.16db.1rodb.32proxy.default	64	16	1	2,048	50,000	320,000	3,200,000
读写分离128G版（16节点1只读）	redis.logic.splitrw.sharding8g.16db.1rodb.32proxy.default	64	16	1	2,048	50,000	320,000	3,200,000
读写分离256G版（32节点1只读）	redis.logic.splitrw.sharding8g.32db.1rodb.64proxy.default	128	32	1	2,048	50,000	500,000	6,400,000
读写分离512G版（32节点1只读）	redis.logic.splitrw.sharding16g.32db.1rodb.64proxy.default	128	32	1	2,048	50,000	500,000	6,400,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和。
- 读写分离实例的总带宽（即表中显示的带宽）上限为2,048 MB/s。达到该上限后，即使选择拥有更多节点数的实例规格，带宽也不会提高。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB/s，则该规格实例的上下行带宽都是10 MB/s。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

5.6.2. 混合存储-标准版（已停售）

本文介绍云数据库Redis企业版混合存储系列标准版架构的规格，包含内存容量、连接数、带宽、QPS参考值等信息。

> **说明** 混合存储型已停止售卖，更多信息，请参见【通知】Redis混合存储型实例停止售卖。推荐选择性能更强，功能更丰富的性能增强型实例。
> 若您已购买混合存储型，您可以通过提交工单迁移数据库实例。

实例规格

规格	InstanceClass （API使用）	CPU核数	每秒新建连接数上限	最大连接数	带宽（MB/s）	QPS参考值
16G内存32G磁盘主从版	redis.amber.master.16g.2x.ext4.default	6	10,000	50,000	48	40,000
16G内存64G磁盘主从版	redis.amber.master.16g.4x.ext4.default	6	10,000	50,000	48	40,000
16G内存128G磁盘主从版	redis.amber.master.16g.8x.ext4.default	6	10,000	50,000	48	40,000
32G内存64G磁盘主从版	redis.amber.master.32g.2x.ext4.default	6	10,000	50,000	48	40,000
32G内存128G磁盘主从版	redis.amber.master.32g.4x.ext4.default	6	10,000	50,000	48	40,000
32G内存256G磁盘主从版	redis.amber.master.32g.8x.ext4.default	6	10,000	50,000	48	40,000
64G内存128G磁盘主从版	redis.amber.master.64g.2x.ext4.default	6	10,000	50,000	48	40,000
64G内存256G磁盘主从版	redis.amber.master.64g.4x.ext4.default	6	10,000	50,000	48	40,000
64G内存512G磁盘主从版	redis.amber.master.64g.8x.ext4.default	6	10,000	50,000	48	40,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和。
- 读写分离实例的总带宽（即表中显示的带宽）上限为2,048 MB/s。达到该上限后，即使选择拥有更多节点数的实例规格，带宽也不会提高。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB/s，则该规格实例的上下行带宽都是10 MB/s。

  > **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

5.6.3. 混合存储-集群版（已停售）

本文介绍云数据库Redis企业版混合存储系列集群版架构的规格，包含内存和磁盘容量、连接数、带宽、QPS参考值等信息。

> **说明** 混合存储型已停止售卖，更多信息，请参见【通知】Redis混合存储型实例停止售卖。推荐选择性能更强，功能更丰富的性能增强型实例。
> 若您已购买混合存储型，您可以通过提交工单迁移数据库实例。

实例规格

规格	InstanceClass（API 使用）	CPU核数	分片数	每秒新建连接数上限	最大连接数（）直连模式	最大连接数（）代理模式	带宽（MB/s）	QPS参考值
64G内存256G磁盘（4分片）	redis.amber.sharding.16g.4db.0rodb.12proxy.4x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
64G内存512G磁盘（4分片）	redis.amber.sharding.16g.4db.0rodb.12proxy.8x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
128G内存512G磁盘（4分片）	redis.amber.sharding.32g.4db.0rodb.12proxy.4x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
128G内存1024G磁盘（4分片）	redis.amber.sharding.32g.4db.0rodb.12proxy.8x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
256G内存1024G磁盘（4分片）	redis.amber.sharding.64g.4db.0rodb.12proxy.4x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
256G内存2048G磁盘（4分片）	redis.amber.sharding.64g.4db.0rodb.12proxy.8x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
128G内存512G磁盘（8分片）	redis.amber.sharding.16g.8db.0rodb.24proxy.4x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
128G内存1024G磁盘（8分片）	redis.amber.sharding.16g.8db.0rodb.24proxy.8x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
256G内存1024G磁盘（8分片）	redis.amber.sharding.32g.8db.0rodb.24proxy.4x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
256G内存2048G磁盘（8分片）	redis.amber.sharding.32g.8db.0rodb.24proxy.8x.ext4.default	48	8	50,000	240,000	500,000	768	240,000

规格	InstanceClass（API 使用）	CPU核数	分片数	每秒新建连接数上限	最大连接数（）直连模式	最大连接数（）代理模式	带宽（MB/s）	QPS参考值
512G内存2048G磁盘（8分片）	redis.amber.sharding.64g.8db.0rodb.24proxy.4x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
512G内存4096G磁盘（8分片）	redis.amber.sharding.64g.8db.0rodb.24proxy.8x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
256G内存1024G磁盘（16分片）	redis.amber.sharding.16g.16db.0rodb.48proxy.4x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
256G内存2048G磁盘（16分片）	redis.amber.sharding.16g.16db.0rodb.48proxy.8x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
512G内存2048G磁盘（16分片）	redis.amber.sharding.32g.16db.0rodb.48proxy.4x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
512G内存4096G磁盘（16分片）	redis.amber.sharding.32g.16db.0rodb.48proxy.8x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
1024G内存4096G磁盘（16分片）	redis.amber.sharding.64g.16db.0rodb.48proxy.4x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
1024G内存8192G磁盘（16分片）	redis.amber.sharding.64g.16db.0rodb.48proxy.8x.ext4.default	96	16	50,000	480,000	500,000	1536	480,000

CPU核数说明

为保障服务稳定运行，系统会保留其中1个CPU用于处理。如果实例为集群或读写分离架构，每个数据分片或每个只读节点均会保留其中1个CPU用于处理后台任务。后台任务

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和，每个分片的带宽限制如下：
  - 社区版：单个分片的内存小于等于1 GB时，每个分片的最大带宽为48 MB/s；单个分片的内存大于1 GB时，每个分片的最大带宽为96 MB/s。
  - 企业版：每个分片的最大带宽为96 MB/s。
- 集群架构的实例使用默认的代理连接地址时，带宽上限为2,048 MB/s。达到该上限后，即使增加分片数量，带宽也不会提高。如需应对业务上的网络超大流量，您可以开通直连访问，具体操作，请参见开通直连访问（仅适用于集群架构）。开启直连访问后：
  - 最大连接数：为单个分片的最大连接数*分片数，社区版单个分片的最大连接数为10,000，企业版单个分片的最大连接数为30,000。
  - 整体带宽限制：为单个分片的最大带宽*分片数，例如128分片的集群实例（单个分片的内存大于1 GB，单个分片的最大带宽为96 MB/s），开启直连后整体带宽为12,288 MB/s。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB，则该规格实例的上下行带宽都是10 MB。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数规则说明

类别	说明
最大连接数	• ：实例最大连接数的上限为500,000，达到该值后，即使增加分片或节点数，最大连接数也不会提高。代理模式 • ：单分片最大连接数的上限为10,000，实例最大连接数的上限为：分片数*10,000。直连模式
每秒新建连接数	每秒新建连接数上限即每秒内可新增的连接数量。例如实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。 **说明** 当您通过直连地址连接实例时，由于绕过了代理节点，连接性能有一定的下降，单个数据分片的每秒新建连接数上限为2,000。例如实例具备4个数据分片，则整体的每秒新建连接数上限为8,000。

5.6.4. 早期已停售规格

云数据库Redis版的部分规格已停止新购，但这些规格的已购实例仍可正常使用。您可以在本章节查看这些已下线规格的连接数限制、带宽、QPS参考值等信息。

社区标准版-同城容灾

规格	InstanceClass （API 使用）	每秒新建连接数上限	连接数上限	带宽（MB/s）	QPS参考值
同城容灾1GB版	redis.logic.sharding.drredissdb1g.1db.0rodb.4proxy.default	10000	10000	10	80000
同城容灾2GB版	redis.logic.sharding.drredissdb2g.1db.0rodb.4proxy.default	10000	10000	16	80000
同城容灾4GB版	redis.logic.sharding.drredissdb4g.1db.0rodb.4proxy.default	10000	10000	24	80000
同城容灾8GB版	redis.logic.sharding.drredissdb8g.1db.0rodb.4proxy.default	10000	10000	24	80000
同城容灾16GB版	redis.logic.sharding.drredissdb16g.1db.0rodb.4proxy.default	10000	10000	32	80000
同城容灾32GB版	redis.logic.sharding.drredissdb32g.1db.0rodb.4proxy.default	10000	10000	32	80000
同城容灾64GB版	redis.logic.sharding.drredissdb64g.1db.0rodb.4proxy.default	10000	10000	48	80000

社区集群版-双副本

规格	InstanceClass （API使用）	分片数	每秒新建连接数上限	连接数上限	带宽（MB/s）	QPS参考值
1G集群版（2分片）	redis.logic.sharding.512m.2db.0rodb.4proxy.default	2	20000	20000	48	200000
1G集群版（4分片）	redis.logic.sharding.256m.4db.0rodb.4proxy.default	4	40000	40000	96	400000
2G集群版（4分片）	redis.logic.sharding.512m.4db.0rodb.4proxy.default	4	40000	40000	96	400000
2G集群版（8分片）	redis.logic.sharding.256m.8db.0rodb.8proxy.default	8	50000	80000	192	800000
4G集群版（8分片）	redis.logic.sharding.512m.8db.0rodb.8proxy.default	8	50000	80000	192	800000
4G集群版（16分片）	redis.logic.sharding.256m.16db.0rodb.16proxy.default	16	50000	160000	384	1600000
8G集群版（16分片）	redis.logic.sharding.512m.16db.0rodb.16proxy.default	16	50000	160000	384	1600000
8G集群版（32分片）	redis.logic.sharding.256m.32db.0rodb.32proxy.default	32	50000	320000	768	3200000
16G集群版（32分片）	redis.logic.sharding.512m.32db.0rodb.32proxy.default	32	50000	320000	768	3200000
16GB集群版	redis.sharding.small.default	8	50000	80000	768	640000
32GB集群版	redis.sharding.mid.default	8	50000	80000	768	640000
64GB集群版	redis.sharding.large.default	8	50000	80000	768	640000
128GB集群版	redis.sharding.2xlarge.default	16	50000	160000	1536	1280000
256GB集群版	redis.sharding.4xlarge.default	16	50000	160000	1536	1280000
512GB集群版	redis.sharding.8xlarge.default	32	50000	320000	2048	2560000

社区集群版-同城容灾

规格	InstanceClass （API使用）	分片数	每秒新建连接数上限	连接数上限	带宽（MB/s）	QPS参考值

规格	InstanceClass （API使用）	分片数	每秒新建连接数上限	连接数上限	带宽（MB/s）	QPS参考值
同城容灾16GB集群版	redis.logic.sharding.drredismdb16g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
同城容灾32GB集群版	redis.logic.sharding.drredismdb32g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
同城容灾64GB集群版	redis.logic.sharding.drredismdb64g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
同城容灾128GB集群版	redis.logic.sharding.drredismdb128g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000
同城容灾256GB集群版	redis.logic.sharding.drredismdb256g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000
同城容灾512GB集群版	redis.logic.sharding.drredismdb512g.32db.0rodb.32proxy.default	32	50000	320000	2048	2560000

带宽计算规则

- 表中的带宽值是整个实例的带宽，即实例中所有分片或节点带宽的总和。
- 读写分离实例的总带宽（即表中显示的带宽）上限为2,048 MB/s。达到该上限后，即使选择拥有更多节点数的实例规格，带宽也不会提高。
- 带宽分别应用于上行带宽和下行带宽，如果某规格的带宽为10 MB/s，则该规格实例的上下行带宽都是10 MB/s。

> **说明** 如果您的实例有突发或计划中的流量高峰，您可以根据需求调整实例的带宽。具体操作，请参见手动调整实例带宽。

- 表中的带宽为Redis实例的内网带宽。外网带宽取决于内网带宽，同时受到Redis实例与客户端之间的网络带宽限制，建议使用内网连接方式，排除外网影响，发挥最大的带宽性能。

连接数计算规则

每秒新建连接数上限即每秒内可新增的连接数量。例如，实例的每秒新建连接数上限为10,000，连接数上限为50,000，实例运行后的第n秒的实际连接数为12,000，则第n+1秒时的连接数最大为只能为22,000（即12,000+10,000）。

常见问题

Q：Redis同城容灾规格都下线了，现在怎么创建有同城容灾能力的实例？

A：Redis同城容灾架构已升级，在创建现有规格的实例时，选择同城容灾可用区即可创建该规格的同城容灾实例。

6.命令支持

6.1. Redis命令支持概览

云数据库Redis存在多个版本、系列和架构，各种类型的实例对于Redis命令的支持度有所不同。根据本章节的导航信息，您可以快速找到云数据库Redis各版本支持的命令和限制使用的命令。

命令支持概览

文档标题	简介
Redis社区版命令支持	云数据库Redis社区版支持多个引擎版本和架构类型，不同的引擎版本和架构类型对Redis命令的支持度有所不同。本文以原生Redis的相关命令为基准，介绍详细的命令支持情况和使用限制，为您的实例选型提供相关参考。
Tair命令限制	Redis企业版兼容大多数的原生Redis命令，为保障服务性能，部分命令的使用受到限制。
Tair扩展数据结构的命令	Redis企业版除支持云Redis社区版的命令以外，还支持下述新的命令： • CAS和CAD命令 • TairString • TairHash • TairGIS • TairBloom • TairDoc • TairTS • TairCpc • TairZset • TairRoaring • TairSearch 说明 • 性能增强型支持以上所有数据结构。 • 持久内存型支持TairString（含CAS和CAD命令）、TairCpc。
阿里云自研的Redis命令	在兼容原生Redis命令之外，云数据库Redis还支持多个自研Redis命令，可以在云Redis集群版或读写分离版中使用，帮助您更方便地管理实例。

不同架构下的命令限制

文档标题	简介
集群架构实例的命令限制	由于部署架构的不同，集群架构的Redis实例在原生Redis命令的支持上有一定的区别。
读写分离实例的命令限制	云数据库Redis版读写分离实例可细分为非集群读写分离和集群读写分离，二者的命令兼容性也有所不同。

6.2. Redis社区版命令支持

云数据库Redis社区版支持多个引擎版本和架构类型，不同的引擎版本和架构类型对Redis命令的支持度有所不同。本文以原生Redis的相关命令为基准，介绍详细的命令支持情况和使用限制，为您的实例选型提供相关参考。

注意事项

- 读写分离架构的实例不支持CLIENT ID命令，同时，在使用某些特定的命令时存在一些限制需要您注意，详情请参见读写分离实例的命令限制。
- 集群架构的实例不支持SWAPDB、CLIENT ID、SORT（BY和GET参数）命令，同时，在使用某些特定的命令时存在一些限制需要您注意，详情请参见集群架构实例的命令限制。

为便于日常管理和运维，集群架构和读写分离架构的实例还支持多个自研的Redis命令，详情请参见阿里云自研的Redis命令。

表格注释说明

为便于浏览和内容表达，本文的表格约定使用下述注释：

- ✔☺表示支持该命令。
- □表示不支持该命令。
- □表示在原生Redis的该版本下，该命令尚未开始支持。例如原生Redis中，TOUCH命令在3.2.1及以上版本才开始支持，表格中的2.8版本下该命令即被标记为□。
- 数字标记①：集群架构实例的在执行该命令时，需要开通直连访问并使用直连地址连接至实例，详情请参见使用直连地址访问Redis实例。通过Proxy节点的连接地址连接至实例时，也兼容支持该命令。
- 数字标记②：为兼容某些客户端框架，执行CONFIG SET命令时仅返回 `OK` ，不会真正地修改参数。

> **说明** 各命令族中的命令，如无特殊备注和说明，默认支持Redis实例的所有架构，即标准架构、集群架构及读写分离架构。关于实例架构的详细介绍，请参见架构信息查询导航。

命令族支持概览

Cluster命令族

> **说明**
> - Cluster命令族的命令不适用于标准架构。
> - 通过Proxy节点的连接地址连接至实例时，会兼容支持部分Cluster命令族的命令，具体为CLUSTER INFO、CLUSTER KEYSLOT、CLUSTER NODES、CLUSTER SLAVES、CLUSTER SLOTS。
> - 云数据库Redis 5.0版自0.1.14版本、云数据库Redis 6.0版自0.1.14版本开始支持READONLY与READWRITE命令，之前版本不支持。

命令	2.8版本	4.0版本	5.0版本	6.0版本
CLUSTER ADDSLOTS	□	□	□	□
CLUSTER BUMPEPOCH	□	□	□	□
CLUSTER COUNT-FAILURE-REPORTS	□	□	□	□
CLUSTER COUNTKEYSINSLOT ①	□	□	□	□

命令	2.8版本	4.0版本	5.0版本	6.0版本
CLUSTER DELSLOTS	□	□	□	□
CLUSTER FAILOVER	□	□	□	□
CLUSTER FLUSHSLOTS	□	□	□	□
CLUSTER FORGET	□	□	□	□
CLUSTER GETKEYSINSLOT	□	□	□	□
CLUSTER INFO ①	✔	✔	✔	✔
CLUSTER KEYSLOT ①	✔	✔	✔	✔
CLUSTER MEET	□	□	□	□
CLUSTER MYID	□	□	□	□
CLUSTER NODES ①	✔	✔	✔	✔
CLUSTER REPLICAS	□	□	□	□
CLUSTER REPLICATE	□	□	□	□
CLUSTER RESET	□	□	□	□
CLUSTER SAVECONFIG	□	□	□	□
CLUSTER SET-CONFIG-EPOCH	□	□	□	□
CLUSTER SETSLOT	□	□	□	□
CLUSTER SLAVES	□	□	□	□
CLUSTER SLOTS	✔	✔	✔	✔
READONLY	□	□	✔	✔
READWRITE	□	□	✔	✔

Connection命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
AUTH	✔	✔	✔	✔
CLIENT CACHING	□	□	□	✔
CLIENT GETNAME	✔	✔	✔	✔
CLIENT GETREDIR	□	□	□	✔
CLIENT ID	□	□	✔	✔
CLIENT KILL	✔	✔	✔	✔
CLIENT LIST	✔	✔	✔	✔
CLIENT PAUSE	□	□	□	□
CLIENT REPLY	□	□	□	□

命令	2.8版本	4.0版本	5.0版本	6.0版本
CLIENT SETNAME	✔	✔	✔	✔
CLIENT TRACKING	✘	✘	✘	✔
CLIENT UNBLOCK	✘	✘	✔	✔
ECHO	✔	✔	✔	✔
HELLO	✘	✘	✘	✔
PING	✔	✔	✔	✔
QUIT	✔	✔	✔	✔
SELECT	✔	✔	✔	✔

Geo命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
GEOADD	✔	✔	✔	✔
GEODIST	✔	✔	✔	✔
GEOHASH	✔	✔	✔	✔
GEOPOS	✔	✔	✔	✔
GEORADIUS	✔	✔	✔	✔
GEORADIUSBYMEMBER	✔	✔	✔	✔

Hashes命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
HDEL	✔	✔	✔	✔
HEXISTS	✔	✔	✔	✔
HGET	✔	✔	✔	✔
HGETALL	✔	✔	✔	✔
HINCRBY	✔	✔	✔	✔
HINCRBYFLOAT	✔	✔	✔	✔
HKEYS	✔	✔	✔	✔
HLEN	✔	✔	✔	✔
HMGET	✔	✔	✔	✔
HMSET	✔	✔	✔	✔
HSCAN	✔	✔	✔	✔
HSET	✔	✔	✔	✔
HSETNX	✔	✔	✔	✔

命令	2.8版本	4.0版本	5.0版本	6.0版本
HSTRLEN	✔	✔	✔	✔
HVALS	✔	✔	✔	✔

HyperLogLog命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
PFADD	✔	✔	✔	✔
PFCOUNT	✔	✔	✔	✔
PFMERGE	✔	✔	✔	✔

Keys命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
DEL	✔	✔	✔	✔
DUMP	✔	✔	✔	✔
EXISTS	✔	✔	✔	✔
EXPIRE	✔	✔	✔	✔
EXPIREAT	✔	✔	✔	✔
KEYS	✔	✔	✔	✔
MIGRATE	□	□	□	□
MOVE	✔	✔	✔	✔
OBJECT	✔	✔	✔	✔
PERSIST	✔	✔	✔	✔
PEXPIRE	✔	✔	✔	✔
PEXPIREAT	✔	✔	✔	✔
PTTL	✔	✔	✔	✔
RANDOMKEY	✔	✔	✔	✔
RENAME	✔	✔	✔	✔
RENAMENX	✔	✔	✔	✔
RESTORE	✔	✔	✔	✔
SCAN	✔	✔	✔	✔
SORT	✔	✔	✔	✔
TOUCH	□	✔	✔	✔
TTL	✔	✔	✔	✔
TYPE	✔	✔	✔	✔

命令	2.8版本	4.0版本	5.0版本	6.0版本
UNLINK	□	✔	✔	✔
WAIT	□	✔	✔	✔

说明 暂不支持通过集群架构的Proxy节点（代理模式）执行WAIT命令，如有需要，您可以通过集群架构的直连地址执行WAIT命令。

Lists命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
BLPOP	✔	✔	✔	✔
BRPOP	✔	✔	✔	✔
BRPOPLPUSH	✔	✔	✔	✔
LINDEX	✔	✔	✔	✔
LINSERT	✔	✔	✔	✔
LLEN	✔	✔	✔	✔
LPOP	✔	✔	✔	✔
LPUSH	✔	✔	✔	✔
LPUSHX	✔	✔	✔	✔
LRANGE	✔	✔	✔	✔
LREM	✔	✔	✔	✔
LSET	✔	✔	✔	✔
LTRIM	✔	✔	✔	✔
RPOP	✔	✔	✔	✔
RPOPLPUSH	✔	✔	✔	✔
RPUSH	✔	✔	✔	✔
RPUSHX	✔	✔	✔	✔

Pub和Sub命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
PSUBSCRIBE	✔	✔	✔	✔
PUBLISH	✔	✔	✔	✔
PUBSUB	✔	✔	✔	✔
PUNSUBSCRIBE	✔	✔	✔	✔
SUBSCRIBE	✔	✔	✔	✔
UNSUBSCRIBE	✔	✔	✔	✔

Scripting命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
EVAL	✔	✔	✔	✔
EVALSHA	✔	✔	✔	✔
SCRIPT DEBUG	✘	✘	✘	✘
SCRIPT EXISTS	✔	✔	✔	✔
SCRIPT FLUSH	✔	✔	✔	✔
SCRIPT KILL	✔	✔	✔	✔
SCRIPT LOAD	✔	✔	✔	✔

Sentinel命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
SENTINEL sentinels	✘	✔	✔	✔
SENTINEL get-master-addr-by-name	✘	✔	✔	✔

Server命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
ACL CAT	✘	✘	✘	✘
ACL DELUSER	✘	✘	✘	✘
ACL GENPASS	✘	✘	✘	✘
ACL GETUSER	✘	✘	✘	✘
ACL HELP	✘	✘	✘	✘
ACL LIST	✘	✘	✘	✘
ACL LOAD	✘	✘	✘	✘
ACL LOG	✘	✘	✘	✘
ACL SAVE	✘	✘	✘	✘
ACL SETUSER	✘	✘	✘	✘
ACL USERS	✘	✘	✘	✘
ACL WHOAMI	✘	✘	✘	✘
BGREWRITEAOF	✘	✘	✘	✘
BGSAVE	✘	✘	✘	✘
COMMAND	✔	✔	✔	✔
COMMAND COUNT	✔	✔	✔	✔
COMMAND GETKEYS	✔	✔	✔	✔

命令	2.8版本	4.0版本	5.0版本	6.0版本
COMMAND INFO	✔	✔	✔	✔
CONFIG GET	✔	✔	✔	✔
CONFIG RESETSTAT	✔	✔	✔	✔
CONFIG REWRITE	✘	✘	✘	✘
CONFIG SET ②	✔	✔	✔	✔
DBSIZE	✔	✔	✔	✔
DEBUG OBJECT	✘	✘	✘	✘
DEBUG SEGFAULT	✘	✘	✘	✘
FLUSHALL	✔	✔	✔	✔
FLUSHDB	✔	✔	✔	✔
INFO	✔	✔	✔	✔
LASTSAVE	✘	✘	✘	✘
LATENCY DOCTOR	✔	✔	✔	✔
LATENCY GRAPH	✔	✔	✔	✔
LATENCY HELP	✘	✘	✔	✔
LATENCY HISTORY	✔	✔	✔	✔
LATENCY LATEST	✔	✔	✔	✔
LATENCY RESET	✔	✔	✔	✔
LOLWUT	✘	✘	✔	✔
MEMORY DOCTOR	✘	✔	✔	✔
MEMORY HELP	✘	✔	✔	✔
MEMORY MALLOC-STATS	✘	✔	✔	✔
MEMORY PURGE	✘	✔	✔	✔
MEMORY STATS	✘	✔	✔	✔
MEMORY USAGE	✘	✔	✔	✔
MODULE LIST	✘	✘	✘	✘
MODULE LOAD	✘	✘	✘	✘
MODULE UNLOAD	✘	✘	✘	✘
MONITOR	✔	✔	✔	✔
PSYNC	✘	✘	✘	✘
REPLICAOF	✘	✘	✘	✘
ROLE	✘	✔	✔	✔

命令	2.8版本	4.0版本	5.0版本	6.0版本
SAVE	□	□	□	□
SHUTDOWN	□	□	□	□
SLAVEOF	□	□	□	□
SLOWLOG	✔	✔	✔	✔
SWAPDB	□	✔	✔	✔
SYNC	□	□	□	□
TIME	✔	✔	✔	✔

Sets命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
SADD	✔	✔	✔	✔
SCARD	✔	✔	✔	✔
SDIFF	✔	✔	✔	✔
SDIFFSTORE	✔	✔	✔	✔
SINTER	✔	✔	✔	✔
SINTERSTORE	✔	✔	✔	✔
SISMEMBER	✔	✔	✔	✔
SMEMBERS	✔	✔	✔	✔
SMISMEMBER	□	□	□	□
SMOVE	✔	✔	✔	✔
SPOP	✔	✔	✔	✔
SRANDMEMBER	✔	✔	✔	✔
SREM	✔	✔	✔	✔
SSCAN	✔	✔	✔	✔
SUNION	✔	✔	✔	✔
SUNIONSTORE	✔	✔	✔	✔

Sorted Sets命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
BZPOPMAX	□	□	✔	✔
BZPOPMIN	□	□	✔	✔
ZADD	✔	✔	✔	✔

命令	2.8版本	4.0版本	5.0版本	6.0版本
ZCARD	✔	✔	✔	✔
ZCOUNT	✔	✔	✔	✔
ZINCRBY	✔	✔	✔	✔
ZINTERSTORE	✔	✔	✔	✔
ZLEXCOUNT	✔	✔	✔	✔
ZPOPMAX	✘	✘	✔	✔
ZPOPMIN	✘	✘	✔	✔
ZRANGE	✔	✔	✔	✔
ZRANGEBYLEX	✔	✔	✔	✔
ZRANGEBYSCORE	✔	✔	✔	✔
ZRANK	✔	✔	✔	✔
ZREM	✔	✔	✔	✔
ZREMRANGEBYLEX	✔	✔	✔	✔
ZREMRANGEBYRANK	✔	✔	✔	✔
ZREMRANGEBYSCORE	✔	✔	✔	✔
ZREVRANGE	✔	✔	✔	✔
ZREVRANGEBYLEX	✔	✔	✔	✔
ZREVRANGEBYSCORE	✔	✔	✔	✔
ZREVRANK	✔	✔	✔	✔
ZSCAN	✔	✔	✔	✔
ZSCORE	✔	✔	✔	✔
ZUNIONSTORE	✔	✔	✔	✔

Streams命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
XACK	✘	✘	✔	✔
XADD	✘	✘	✔	✔
XCLAIM	✘	✘	✔	✔
XDEL	✘	✘	✔	✔
XGROUP	✘	✘	✔	✔
XINFO	✘	✘	✔	✔
XLEN	✘	✘	✔	✔

命令	2.8版本	4.0版本	5.0版本	6.0版本
XPENDING	□	□	✔	✔
XRANGE	□	□	✔	✔
XREAD	□	□	✔	✔
XREADGROUP	□	□	✔	✔
XREVRANGE	□	□	✔	✔
XTRIM	□	□	✔	✔

Strings命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
APPEND	✔	✔	✔	✔
BITCOUNT	✔	✔	✔	✔
BITFIELD	✔	✔	✔	✔
BITOP	✔	✔	✔	✔
BITPOS	✔	✔	✔	✔
DECR	✔	✔	✔	✔
DECRBY	✔	✔	✔	✔
GET	✔	✔	✔	✔
GETBIT	✔	✔	✔	✔
GETRANGE	✔	✔	✔	✔
GETSET	✔	✔	✔	✔
INCR	✔	✔	✔	✔
INCRBY	✔	✔	✔	✔
INCRBYFLOAT	✔	✔	✔	✔
MGET	✔	✔	✔	✔
MSET	✔	✔	✔	✔
MSETNX	✔	✔	✔	✔
PSETEX	✔	✔	✔	✔
SET	✔	✔	✔	✔
SETBIT	✔	✔	✔	✔
SETEX	✔	✔	✔	✔
SETNX	✔	✔	✔	✔
SETRANGE	✔	✔	✔	✔

命令	2.8版本	4.0版本	5.0版本	6.0版本
STRALGO	□	□	□	✔
STRLEN	✔	✔	✔	✔

Transaction命令族

命令	2.8版本	4.0版本	5.0版本	6.0版本
DISCARD	✔	✔	✔	✔
EXEC	✔	✔	✔	✔
MULTI	✔	✔	✔	✔
UNWATCH	✔	✔	✔	✔
WATCH	✔	✔	✔	✔

6.3. Tair扩展数据结构的命令

阿里云Tair实例（云数据库Redis企业版）除了支持云Redis社区版命令，还集成了多个扩展数据结构，可帮助您简化开发流程，提升数据处理效率。

扩展数据结构列表

扩展数据结构	说明
CAS和CAD命令	为增强Redis String的功能而开发的命令，可以实现简洁高效的Redis分布式锁。
TairString	TairString是一种带版本号的String类型数据结构，TairString除包含Key和Value外，还携带了版本（version）信息。除此之外，TairString在Redis String加减功能的基础上支持了边界设置，可以将INCRBY、INCRBYFLOAT的结果限制在一定的范围内，超出范围则提示错误。
TairHash	TairHash是一种Hash类型的数据，不但和原有的Redis Hash一样支持丰富的数据接口和高处理性能，还支持为field设置过期时间和版本，简化业务开发流程。通过高效的Active Expire算法，可以在不对响应时间造成明显影响的前提下，更高效的完成对field的超时判断和删除。
TairGIS	TairGIS是一种使用R-Tree做索引，支持地理信息系统GIS（Geographic Information System）相关接口的数据结构。Redis的原生GEO命令使用1D索引，主要用于点的查询，TairGIS使用2D索引，支持点、线、面的查询，适合判断相交或包含关系，功能更加强大。
TairBloom	TairBloom是一种可动态扩容的布隆过滤器，完全兼容RedisBloom模块的命令，具有动态扩容的能力，相对传统实现方式消耗内存更低，可在扩容的同时维持误判率的稳定，适合需要高效判断大量数据是否存在且允许一定误判率的业务场景。
TairDoc	TairDoc是一种文档类型的数据结构，支持JSON标准，完全兼容ReJSON模块的命令，同时，TairDoc数据以二进制树的方式存储，支持对JSON中子元素的快速访问。
TairTS	TairTS是基于Redis Module开发的时序数据结构，提供低时延、高并发的内存读写访问，及快速的过滤、聚合查询功能，集存储与计算为一体，在简化了处理时序数据流程的同时，大幅度提高了性能。
TairCpc	TairCpc是基于CPC（Compressed Probability Counting）压缩算法开发的数据结构，支持仅占用很小的内存空间对采样数据进行高性能计算。
TairZset	TairZset可实现任意维度的double类型的分值排序，提升数据处理效率，且客户端适配简易，无需任何编解码封装，解决原生Sorted Set（也称Zset）只支持1个double类型的分值排序的局限性。
TairRoaring	TairRoaring是基于Tair引擎的Roaring Bitmap实现，通过2层索引和引入多种动态容器（Container），同时使用了包括SIMD instructions、Vectorization、PopCnt算法在内的等多种工程优化，提供更低的内存占用及更高的集合计算效率。

扩展数据结构	说明
TairSearch	TairSearch是基于Redis module全自研（不基于Lucene等开源搜索库）的全文搜索模块，采用和Elasticsearch相似（ES-LIKE）的查询语法。

> 说明
> - 性能增强型支持以上所有数据结构。
> - 持久内存型支持TairString（含CAS和CAD命令）、TairCpc。

其它命令

除上述特有的命令外，Tair实例支持云数据库Redis社区版的所有命令，更多信息请参见Redis社区版命令支持。

6.4. 阿里云自研的Redis命令

在兼容原生Redis命令之外，云数据库Redis版还支持多个自研的Redis命令，可以在云Redis集群版或读写分离版中使用，帮助您更方便地管理实例。

前提条件

通过代理模式（Proxy）访问Redis实例。

自研命令说明

- info key命令：查询key所属的slot和db。Redis原生的info命令中最多可以带一个可选的section（`info [section]`）。目前云数据库Redis版的集群实例，部分命令限制所有key必须在同一个slot中，`info key`命令方便用户查询某些key是否在同一个slot或db节点中。用法如下：

  ```
  127.0.0.1:6379> info key test_key
  slot:15118 node_index:0
  ```

  > 注意
  > - 旧版本Redis实例中可能出现`info key`显示出来的`node index`和实例拓扑图的`node index`不一致，最新版本已经修复。如果您的实例版本较旧，请先升级，方法请参见升级小版本。
  > - `info key`显示的node是指集群规格下后端的物理节点，和select命令中的db不是同一个概念。

- iinfo命令：类似info命令，用于在指定的Redis节点上执行info命令。用法如下：

  ```
  iinfo db_idx [section]
  ```

  其中，db_idx的范围是[0, nodecount]，nodecount可以通过info命令获取，section的用法与官方info命令中的section一致。要了解某个Redis节点的info可以使用iinfo命令或者从控制台上查看实例拓扑图。

- riinfo命令：和iinfo命令类似，但只能在读写分离的模式下使用。用法中增加了一个readonly slave的idx，用于指定在第几个readonly slave上执行info命令。在读写分离集群中可以用来在指定readonly slave上执行info命令。如果在非读写分离集群中使用，会返回错误。用法如下：

  ```
  riinfo db_idx ro_slave_idx [section]
  ```

- iscan命令：在集群模式下可以在指定的db节点上执行scan命令。在scan命令的基础上扩展了一个参数用于指定db_idx，db_idx的范围是[0, nodecount]，nodecount可以通过info命令获取或者从控制台上查看实例拓扑图。用法如下：

  ```
  iscan db_idx cursor [MATCH pattern] [COUNT count]
  ```

- imonitor命令：和iinfo、iscan类似，在monitor的基础上新增一个参数指定monitor执行的db_idx，db_idx的范围是[0, nodecount)，nodecount可以通过info命令获取或者从控制台上查看实例拓扑图。用法如下：

  ```
  imonitor db_idx
  ```

- rimonitor命令：和riinfo类似，用于读写分离场景下，在指定的shard里的指定只读从库上执行monitor命令。用法如下：

  ```
  rimonitor db_idx ro_slave_idx
  ```

> **说明** imonitor和rimonitor请用telnet连接后执行，如需退出imonitor、rimonitor，请使用**quit**命令。

6.5. Tair命令限制

阿里云Tair实例（云数据库Redis企业版）兼容大多数的原生Redis命令，为保障服务性能，部分命令的使用受到限制。

Tair介绍

随着互联网的高速发展，业务场景变得越来越丰富和复杂，Redis企业版作为一个高可用、高性能的分布式NoSQL数据库，从访问延时、持久化需求、整体成本这三个核心维度考量，基于DRAM、NVM和ESSD云盘存储介质，推出了多种不同形态的产品，为您提供更强的性能、更多的数据结构和更灵活的存储方式，满足不同场景下的业务需求。

关于Tair实例各形态产品的详细信息，请参见：

- 性能增强型
- 持久内存型
- 容量存储型
- 混合存储型（已停售）

性能增强型

暂无限制。性能增强型实例除支持云数据库Redis社区版的命令以外，还支持一些新的命令，详情请参见Tair扩展数据结构的命令。

持久内存型

命令族	不支持的命令
Keys（键）	RENAME、RENAMENX、MOVE
Server（数据库管理）	SWAPDB
Streams（流）	XACK、XADD、XCLAIM、XDEL、XGROUP、XINFO、XLEN、XPENDING、XRANGE、XREAD、XREADGROUP、XREVRANGE、XTRIM **说明** 持久内存型自1.2.3版本开始支持Streams（流），之前版本不支持。

在持久内存型实例中，执行部分更新命令的性能会随着原始Key的容量增大而下降，当原始Key的容量为MB级别时，引擎内延迟将显著增加，涉及的命令为：SETRANGE、SETBIT、APPEND命令、BITFIELD相关更新命令、Geo相关更新命令以及HyperLogLog相关更新命令。如有以上需求，推荐使用性能增强型实例。

> **说明** 如果大Key是作为Bitmap使用，推荐使用性能增强型的TairRoaring Module，在提供高性能计算能力的同时节省存储空间，更多信息请参见TairRoaring。

容量存储型

命令族	不支持的命令
Geo（地理位置）	GEOADD、GEODIST、GEOHASH、GEOPOS、GEORADIUS、GEORADIUS_RO、GEORADIUSBYMEMBER、GEORADIUSBYMEMBER_RO
Hyperloglog	PFADD、PFDEBUG、PFCOUNT、PFMERGE、PFSELFTEST
Keys（键）	RENAME、RENAMENX、MOVE、OBJECT、SORT、TOUCH
Lists（列表）	BRPOP、BLPOP、BRPOPLPUSH
Scripting（Lua脚本）	EVAL、EVALSHA、SCRIPT DEBUG、SCRIPT EXISTS、SCRIPT FLUSH、SCRIPT KILL、SCRIPT LOAD
Strings（字符串）	BITCOUNT、BITFIELD、BITOP、BITPOS、GETBIT、SETBIT

命令族	不支持的命令
Server（数据库管理）	MEMORY DOCTOR、MEMORY HELP、MEMORY MALLOC-STATS、MEMORY PURGE、MEMORY STATS、MEMORY USAGE、SWAPDB
Transactions（事务）	DISCARD、EXEC、MULTI、UNWATCH、WATCH

混合存储型

命令族	不支持的命令
Keys（键）	RENAME、RENAMENX、MOVE、SORT（STORE选项）
Lists（列表）	LINSERT、LREM
Server（数据库管理）	SWAPDB
Scripting（Lua脚本）	SCRIPT DEBUG、SCRIPT LOAD

6.6. 集群架构实例的命令限制

由于部署架构的不同，相对标准架构来说，集群架构的实例在原生Redis命令的支持上有一定的区别，本文介绍集群架构实例的命令使用限制。

支持的命令

- 集群架构的Redis社区版实例：详情请参见Redis社区版命令支持。
- 集群架构的Redis企业版实例：Redis企业版拥有多个形态，不同形态的产品对命令的支持有所区别，详情请参见Tair命令限制。

> **说明** 集群架构的Redis实例支持直连模式和代理模式，如需通过JedisCluster使用multi或exec方法，请使用代理模式。

不支持的命令

- SWAPDB
- CLIENT ID
- SORT（BY和GET参数）

受限的命令

> **说明** 如需在集群架构实例中执行下述受限制的命令，请使用hash tag确保命令所要操作的key都分布在1个hash slot中，hash tag的详细用法请参见Redis官方文档。

命令族	具体命令
HyperLogLog	PFMERGE、PFCOUNT
Keys	RENAME、RENAMENX、SORT
Lists	RPOPLPUSH、BRPOP、BLPOP、BRPOPLPUSH
Scripting	EVAL、EVALSHA、SCRIPT EXISTS、SCRIPT FLUSH、SCRIPT KILL、SCRIPT LOAD
Strings	MSETNX
Transaction	DISCARD、EXEC、MULTI、UNWATCH、WATCH

SELECT命令的限制

连接模式	说明
代理模式	支持，需要注意的是某些开源Redis客户端（例如stackExchange.redis）由于其误判导致无法正常执行SELECT命令，您可以先将cluster_compat_enable参数设置为*0*（即关闭原生Redis Cluster语法兼容），然后重启客户端应用后再进行尝试。具体操作，请参见设置实例参数。您也可以更其他支持该命令的客户端，例如Jedis客户端。
直连模式	不支持，主流客户端（例如Jedis客户端）的自身限制导致。

集群架构中Lua脚本的限制

警告 由于集群架构对Lua脚本的使用存在一定的限制，在将实例变更至集群架构时（通过变更实例配置实现），Lua脚本可能因脚本内容不符合限制而发生丢失，请务必提前备份。

Redis Cluster对使用Lua脚本增加了一些限制，云数据库Redis集群版在这个基础上存在如下额外限制：

说明 如果发现无法执行Eval的相关命令，例如提示 `ERR command eval not support for normal user` ，请将实例的小版本升级至最新。具体操作，请参见升级小版本。

- 所有key必须在一个slot上，否则返回错误信息：

  ```
  -ERR eval/evalsha command keys must be in same slot\r\n
  ```

  说明 您可以通过CLUSTER KEYSLOT命令获取目标key的哈希槽（hash slot）。

- 对单个节点执行SCRIPT LOAD命令时，不保证将该Lua脚本存入至其他节点中。
- 不支持发布订阅命令，包括**PSUBSCRIBE**、**PUBSUB**、**PUBLISH**、**PUNSUBSCRIBE**、**SUBSCRIBE**和**UNSUBSCRIBE**。
- 不支持UNPACK函数。

若您能够在代码中确保所有操作都在相同slot（如果不能保障这一点，执行会出错），且希望打破Redis集群的Lua限制，可以在控制台将script_check_enable修改为*0*，则后端不会对脚本进行校验，但仍需要使用KEYS数组至少传递一个key，供代理节点执行路由转发。具体操作，请参见设置实例参数。

代理模式的额外限制

- 所有key都应该由KEYS数组来传递，**redis.call/pcall**中调用的Redis命令，key的位置必须是KEYS array（不能使用Lua变量替换KEYS），否则直接返回错误信息：

  ```
  -ERR bad lua script for redis cluster, all the keys that the script uses should be passed using the KEYS ar
  ray\r\n
  ```

  正确与错误命令示例如下：

  ```
  # 本示例的准备工作需执行如下命令
  SET foo foo_value
  SET {foo}bar bar_value
  # 正确示例
  EVAL "return redis.call('mget', KEYS[1], KEYS[2])" 2 foo {foo}bar
  # 错误示例
  EVAL "return redis.call('mget', KEYS[1], '{foo}bar')" 1 foo
  EVAL "return redis.call('mget', KEYS[1], ARGV[1])" 1 foo {foo}bar
  ```

- 调用必须要带有key，否则直接返回错误信息：

  ```
  -ERR for redis cluster, eval/evalsha number of keys can't be negative or zero\r\n
  ```

  正确与错误命令示例如下：

  ```
  # 正确示例
  EVAL "return redis.call('get', KEYS[1])" 1 foo
  # 错误示例
  EVAL "return redis.call('get', 'foo')" 0
  ```

- 不支持在MULTI、EXEC事务中使用EVAL、EVALSHA、SCRIPT系列命令。

> **说明** 若您需要使用代理模式下受限的部分功能，您可以尝试开通使用云数据库Redis集群版的直连模式。但是由于云数据库Redis集群版在迁移、变配时都会通过proxy代理迁移数据，直连模式下不符合代理模式的Lua脚本会迁移、变配失败。建议您在直连模式下使用Lua脚本时应尽可能符合代理模式下的限制规范，避免后续Lua脚本迁移、变配失败。

其他限制

- 执行CLIENT LIST命令会列出所有连接到该代理节点的连接信息。返回结果解释如下：
  - `id`、`age`、`idle`、`addr`、`fd`、`name`、`db`、`multi`、`omem`、`cmd`字段和原生Redis的含义一致。
  - `sub`、`psub`在代理节点上没有区分，统一为1或0。
  - `qbuf`、`qbuf-free`、`obl`和`oll`字段目前没有具体意义。
- CLIENT KILL命令目前支持的形式为：`client kill ip:port`和`client kill addr ip:port`。
- 对于集群架构实例，在代理模式下，为了兼容主从架构，事务会被拆分成多个子事务执行，此时无法保证事务的原子性；在直连模式下，与开源Redis Cluster行为一致，即严格要求事务所操作的key均在同一slot。

  > **注意** 在代理模式下，当使用WATCH或事务内有多key命令（如MSET）时，事务不会被拆分，严格要求事务所操作的Key均在同一Slot，此时可以保证事务的原子性。

  - 多key命令包括：DEL、SORT、MGET、MSET、BITOP、EXISTS、MSETNX、RENAME、RENAMENX、BLPOP、BRPOP、RPOPLPUSH、BRPOPLPUSH、SMOVE、SUNION、SINTER、SDIFF、SUNIONSTORE、SINTERSTORE、SDIFFSTORE、ZUNIONSTORE、ZINTERSTORE、PFMERGE、PFCOUNT。
  - 不允许在事务中使用的命令为：WATCH、UNWATCH、RANDOMKEY、KEYS、SUBSCRIBE、UNSUBSCRIBE、PSUBSCRIBE、PUNSUBSCRIBE、PUBLISH、PUBSUB、SCRIPT、EVAL、EVALSHA、SCAN、ISCAN、DBSIZE、ADMINAUTH、AUTH、PING、ECHO、FLUSHDB、FLUSHALL、MONITOR、IMONITOR、RIMONITOR、INFO、IINFO、RIINFO、CONFIG、SLOWLOG、TIME、CLIENT。

6.7. 读写分离实例的命令限制

云数据库Redis版读写分离实例可细分为非集群读写分离和集群读写分离，二者的命令限制也有所不同。

实例架构	架构说明	命令限制
非集群读写分离	非集群读写分离实例即单分片的读写分离实例，架构中包含1个由master和replica组成的读写分片，以及1个、3个或5个只读副本。	命令支持特性与Redis标准版实例相同，详情请参见Redis社区版命令支持。
集群读写分离	集群读写分离实例，又称为读写分离集群版，架构中包含若干个由master和replica组成的读写分片，且每个读写分片都有1个与之对应的只读副本。	命令支持特性与Redis集群版实例相同，详情请参见集群架构实例的命令限制。

为便于日常管理和运维，集群架构和读写分离架构的实例还支持多个自研的Redis命令，详情请参见阿里云自研的Redis命令。

7.版本说明

7.1. 大版本特性说明

7.1.1. Redis 7.0新特性说明

Redis 7.0版本发布多个新特性并进行多项改进优化。

更新说明

- 新增Function自定义函数库，函数库支持持久化与可复制。
- Lua脚本（脚本本身代码）不再支持持久化和复制，仅对命令执行结果进行持久化和复制。
- ACL v2版本，支持Key级别的权限控制。
- ACL支持对Pub/Sub channel的权限控制。
- 支持Multi-Part AOF。
- 支持Client-Eviction。
- 支持Sharded-Pub/Sub。
- 支持命令执行耗时直方图。
- 支持子命令级别的性能统计。
- Ziplist编码替换为Listpack编码。
- 支持Global Replication Buffer。

7.1.2. Redis 6.0新特性说明

Redis 6.0版本在一系列关键领域进行了改进，本章节介绍该版本新增的主要特性。

更新说明

- 模块系统新增多个API。
- 支持新的Redis协议：RESP3。
- 服务端支持多模式的客户端缓存。
- 支持多线程IO。
- 副本中支持无盘复制（diskless replication）。
- Redis-benchmark新增了Redis集群模式。
- 支持重写Systemd。
- 支持Disque模块。

7.1.3. Redis 5.0新特性说明

云数据库Redis 5.0版本大幅度优化内核，运行更加稳定，同时新增Stream、账号管理、审计日志等多种特性，满足您更多场景下的使用需求。

更新说明

- 新的数据类型：流数据（Stream）。详细说明请参见Redis Streams。
- 新增创建与管理账号功能。
- 新增日志管理功能，支持审计日志、运行日志和慢日志，您可以通过日志管理查询读写操作、敏感操作（如KEYS、FLUSHALL）和管理类命令的使用记录以及慢日志。
- 新增基于快照的离线全量Key分析功能。
- 新的定时器（Timers）、集群（Cluster）和字典（Dictionary）模块的API。
- RDB中增加LFU和LRU信息。
- 集群管理器从Ruby（redis-trib.rb）移植到了redis-cli中的C语言代码。
- 新增有序集合（Sorted Set）命令ZPOPMIN、ZPOPMAX、BZPOPMIN和BZPOPMAX。
- 升级Active Defragmentation至v2版本。
- 增强HyperLogLog的实现。

- 优化内存统计报告。
- 为许多有子命令的命令增加了HELP子命令。
- 提高了客户端频繁连接和断开连接时的性能表现。
- 升级Jemalloc至5.1版本。
- 新增命令CLIENT ID和CLIENT UNBLOCK。
- 新增了为艺术而生的LOLWUT命令。
- 弃用slave术语（需要API向后兼容的情况例外）。
- 对网络层进行了多处优化。
- 进行了一些Lua相关的改进。
- 新增动态HZ（Dynamic HZ）以平衡空闲CPU使用率和响应性。
- 对Redis核心代码进行了重构并在许多方面进行了改进。

7.1.4. Redis 4.0新功能介绍

云数据库Redis 4.0版是以社区4.0引擎为基础，合入了大量阿里云开发的特性，并且修复了许多bug后全新推出的售卖版本。除了拥有Redis 2.8引擎所具备的所有优势之外，还带来了一些新的功能特性。

Lazyfree

Redis 4.0的Lazyfree机制可以避免 `del` 、 `flushdb` 、 `flushall` 、 `rename` 等命令引起的redis-server阻塞，提高服务稳定性，详情如下。

unlink

在Redis 4.0之前，Redis执行 `del` 命令，只有在释放掉key的所有内存以后才会返回 `OK` 。如果key比较大（比如说一个hash里有1000万条数据），其他连接需要等待较长时间。为了兼容已有的 `del` 语义，Redis 4.0引入 `unlink` 命令，效果以及用法和 `del` 完全一样，但内存释放动作放到后台线程中执行。

```
UNLINK key [key ...]
```

flushdb/flushall

`flushdb/flushall` 在 Redis 4.0中引入了新选项，可以指定是否使用Lazyfree的方式来清空整个内存。

```
FLUSHALL [ASYNC]
FLUSHDB [ASYNC]
```

rename

执行 `rename oldkey newkey` 时，如果newkey已经存在，Redis会先删除已经存在的newkey，这也会引发上面提到的删除大key问题。如果想让Redis在这种场景下也使用lazyfree的方式来删除，您可以在控制台上打开如下配置：

```
lazyfree-lazy-server-del yes/no
```

> **说明**
> 该参数配置在控制台中暂未开放。

淘汰或者逐出数据

有些用户对数据设置过期时间，依赖Redis的淘汰机制去删除已经过期的数据，这同样也存在上面提到的问题：淘汰某个大key会导致进程CPU出现抖动。Redis 4.0提供了两个配置，可以让Redis在淘汰或者逐出数据时也使用lazyfree的方式。

```
lazyfree-lazy-eviction yes/no
lazyfree-lazy-expire yes/no
```

新增命令

swapdb

`swapdb` 命令会交换两个db的数据， `swapdb` 执行之后用户连接db无需再执行 `select` ，即可看到新的数据。

```
127.0.0.1:6379> select 0
OK
127.0.0.1:6379> set key value0
OK
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> set key value1
OK
127.0.0.1:6379[1]> swapdb 0 1
OK
127.0.0.1:6379[1]> get key
"value0"
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> get key
"value1"
```

zlexcount

`zlexcount` 命令用于sorted set中，和 `zrangebylex` 类似，不同的是 `zrangebylex` 返回member，而 `zlexcount` 是返回符合条件的member个数。

memory

Redis 4.0之前只能通过 `info memory` 来了解Redis内部有限的内存信息，Redis 4.0提供了 `memory` 命令，帮助用户全面了解Redis的内存状态。

```
127.0.0.1:6379> memory help
1) "MEMORY DOCTOR                        - Outputs memory problems report"
2) "MEMORY USAGE <key> [SAMPLES <count>] - Estimate memory usage of key"
3) "MEMORY STATS                         - Show memory usage details"
4) "MEMORY PURGE                         - Ask the allocator to release memory"
5) "MEMORY MALLOC-STATS                  - Show allocator internal stats"
```

- `memory usage`
  `usage` 子命令可以查看某个key在Redis内部实际占用多少内存。

  > 注意
  > - 不光key、value需要占用内存，Redis管理这些数据还需要一部分内存。
  > - 对于hash、list、set、sorted set这些类型的key，结果是采样计算的，可以通过 `SAMPLES` 来控制采样数量。

- `memory stats`

```
27.0.0.1:6379> memory stats
       1) "peak.allocated"    // Redis从启动到现在，历史最多使用过多少内存
       2) (integer) 423995952
       3) "total.allocated"    //当前使用内存
       4) (integer) 11130320
       5) "startup.allocated"    //Redis启动初始化以后占用内存
       6) (integer) 9942928
       7) "replication.backlog"    //主从复制断开重连时会用到，默认10MB
       8) (integer) 1048576
       9) "clients.slaves"    // 主从复制用到的内存
      10) (integer) 16858
      11) "clients.normal"    //普通用户客户端的读写缓冲区
      12) (integer) 49630
      13) "aof.buffer"    //aof持久化使用的缓存和aofrewrite时产生的缓存之和
      14) (integer) 3253
      15) "db.0"    //每个db的元数据所占用内存
      16) 1) "overhead.hashtable.main"
          2) (integer) 5808
          3) "overhead.hashtable.expires" //管理带过期时间的数据所额外消耗内存
          4) (integer) 104
      17) "overhead.total"    //上面提到的各项内存消耗之和
      18) (integer) 11063904
      19) "keys.count"    //当前存储的key的总量
      20) (integer) 94
      21) "keys.bytes-per-key"    //当前内存中平均每个key大小
      22) (integer) 12631
      23) "dataset.bytes"        //用户数据所占用内存(= 总内存 - Redis元数据所占内存)
      24) (integer) 66416
      25) "dataset.percentage"    //100 * dataset.bytes / (total.allocated - startup.allocated)
      26) "5.5934348106384277"
      27) "peak.percentage"    // 100 * total.allocated / peak_allocated
      28) "2.6251003742218018"
      29) "fragmentation"    //内存碎片率
      30) "1.1039986610412598"
```

- `memory doctor`
  主要用于给一些诊断建议，提前发现潜在问题。

```
Peak memory: peak.allocated/total.allocated > 1.5，此时内存碎片率可能比较高
  High fragmentation: fragmentation > 1.4，此时碎片率比较高
  Big slave buffers: 每个slave缓冲区的平均内存超过10MB，原因可能是master写入流量过高
  Big client buffers: 普通客户端缓冲区的平均内存超过200KB，原因可能是pipeline使用不当或者Pub/Sub客户端处理消息不及时导致
```

- `malloc stats` & `malloc purge`
  这两个命令用于操作jemalloc，只在使用jemalloc的时候才有效。

LFU机制与hotkey

Redis 4.0新增了allkey-lfu和volatile-lfu两种数据逐出策略，同时还可以通过 `object` 命令来获取某个key的访问频度。

```
object freq user_key
```

基于LFU机制，用户可以使用 `scan + object freq` 来发现热点key，当然Redis也一起发布了更好用的工具redis-cli，使用示例如下。

```
$./redis-cli --hotkeys
# Scanning the entire keyspace to find hot keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).
[00.00%] Hot key 'counter:000000000002' found so far with counter 87
[00.00%] Hot key 'key:000000000001' found so far with counter 254
[00.00%] Hot key 'mylist' found so far with counter 107
[00.00%] Hot key 'key:000000000000' found so far with counter 254
[45.45%] Hot key 'counter:000000000001' found so far with counter 87
[45.45%] Hot key 'key:000000000002' found so far with counter 254
[45.45%] Hot key 'myset' found so far with counter 64
[45.45%] Hot key 'counter:000000000000' found so far with counter 93
-------- summary -------
Sampled 22 keys in the keyspace!
hot key found with counter: 254 keyname: key:000000000001
hot key found with counter: 254 keyname: key:000000000000
hot key found with counter: 254 keyname: key:000000000002
hot key found with counter: 107 keyname: mylist
hot key found with counter: 93  keyname: counter:000000000000
hot key found with counter: 87  keyname: counter:000000000002
hot key found with counter: 87  keyname: counter:000000000001
hot key found with counter: 64  keyname: myset
```

7.2. 小版本发布日志

7.2.1. Redis企业版小版本发布日志

为提升用户体验，云数据库Redis会不定期地发布小版本，用于丰富云产品功能或修复已知缺陷。您可以参阅本文了解Redis企业版小版本的更新说明，选择在业务低峰期升级实例的小版本。

如何查询或升级实例的小版本

您可以通过控制台查看当前的小版本，具体操作及升级小版本的其注意事项，请参见升级小版本。

查看小版本

> **注意**
>
> - 系统会自动检测实例的小版本，如果**小版本升级**按钮不存在或处于无法单击的状态，表示该实例已经是最新的小版本。
> - 由于各地域版本发布进度可能有所差异，小版本发布情况以当前控制台显示为准。

企业版介绍

随着互联网的高速发展，业务场景变得越来越丰富和复杂，Redis企业版作为一个高可用、高性能的分布式NoSQL数据库，从访问延时、持久化需求、整体成本这三个核心维度考量，基于DRAM、NVM和ESSD云盘存储介质，推出了多种不同形态的产品，为您提供更强的性能、更多的数据结构和更灵活的存储方式，满足不同场景下的业务需求。

Redis企业版产品类型	特性

Redis企业版产品类型	特性
性能增强型	• 采用多线程模型：性能约为同规格社区版实例的3倍。 • 支持多种增强型数据结构模块（modules），包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，使业务无需再关心存储的结构和时效性，能够极大提升业务开发效率。 • 超高兼容性：100%兼容原生Redis，无需修改业务代码。 • 支持诸多企业级特性：通过数据闪回按时间点恢复数据、代理查询缓存、全球多活等。
持久内存型	• 超高性价比：相同容量下对比Redis社区版，价格降低30%左右，性能可达原生Redis的90%。 • 支持增强型数据结构模块（modules）：TairString（含CAS和CAD命令）、TairCpc。 • 掉电数据不丢失：强大的命令级持久化保障，每个写操作持久化成功后返回，可将其作为内存数据库（非缓存）使用。 • 大规格优化：解决大规格下执行AOF重写调用fork引起的延时抖动等问题。 • 高兼容性：兼容绝大部分原生Redis的数据结构和命令。
容量存储型	容量存储型基于云盘ESSD研发，兼容Redis核心数据结构与接口，可提供大容量、低成本、强持久化的数据库服务。容量存储型在降低成本和提升数据可靠性的同时，也解决了原生Redis固有的因fork而预留部分内存的问题。适用于兼容Redis、需要大容量且较高访问性能的温冷数据存储场景。
混合存储型（停止售卖）	混合存储型采用内存加磁盘的存储模式，能够在业务高峰期后对冷热数据进行弹性分离，既保障了热数据的内存访问速度，又提供了远超社区Redis的存储容量，实现了性能与成本的平衡。 **说明** 混合存储型已停止售卖，更多信息，请参见【通知】Redis混合存储型实例停止售卖。推荐选择性能更强，功能更丰富的性能增强型实例。 若您已购买混合存储型，您可以通过提交工单迁移数据库实例。

更新级别说明

- LOW：一般级别，包含日常新功能升级（例如新增某个功能）。
- MEDIUM：推荐级别， 包含功能模块优化类的升级（例如优化了某个功能）。除此以外，还包含了LOW级别所包含的更新内容。
- HIGH：重要级别，包含影响稳定性或安全性的重要升级（例如修复某个漏洞或缺陷）。除此以外，还包含LOW和MEDIUM级别所包含的更新内容。

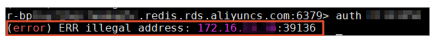
企业版（性能增强型）

小版本号	更新级别	发布日期	类型	说明
1.8.9	LOW	2022-06-07	功能优化	• 提升性能增强型实例开通直连访问后的服务稳定性。
1.8.8	LOW	2022-06-01	功能优化	• 更新TairSearch，提升联合查询效率。
1.8.5	HIGH	2022-05-24	安全加固	• 提升性能增强型集群架构实例开通直连访问后，变配的稳定性。 • 支持TairSearch聚合功能，并提升写文档效率。
1.8.4	LOW	2022-05-17	功能优化	• 提升TairSearch的稳定性和执行效率。
1.8.3	LOW	2022-04-28	功能优化	• 增强TairTS的稳定性。
1.8.2	HIGH	2022-04-22	安全加固	• 修复集群架构实例开通直连访问后，变配过程中迁移大Key概率性失败的问题。
			功能优化	• 优化TairSearch查询效率。

小版本号	更新级别	发布日期	类型	说明
1.8.1	LOW	2022-04-20	功能优化	• 发布TairRoaring V2.2，详情请参见TairRoaring。
1.7.28	LOW	2022-03-24	功能优化	• 延时统计直方图（Latency）支持统计Tair module命令，详情请参见延时洞察。
1.7.27	LOW	2022-03-11	功能优化	• 发布TairRoaring V2，详情请参见TairRoaring。 • 发布TairSearch。
1.7.20	MEDIUM	2022-01-04	功能优化	• 优化TairRoaring性能。
1.7.17	LOW	2021-11-29	功能优化	• 修复异常情况下集群实例重启失败的问题。 说明 仅发布云盘版。
1.7.16	LOW	2021-11-24	功能优化	• 修复实时Key分析功能遗漏统计Spop命令的问题。
1.7.12	MEDIUM	2021-10-26	功能优化	• 修复集群变配过程中慢日志过多的问题，增强稳定性。
1.7.11	MEDIUM	2021-10-15	功能优化	• 在Info结果中增加TDE信息。 • 增强稳定性。
1.7.9	LOW	2021-10-13	功能优化	• 增强了集群变配时槽（slot）无感迁移的自治能力。
1.7.8	MEDIUM	2021-09-26	功能优化	• TairGis数据结构优化Raycast算法问题，解决GIS.CONTAINS命令搜索部分case不准确的问题。
1.7.7	MEDIUM	2021-09-13	功能优化	• 支持TairRoaring。 • 增强稳定性。
1.7.6	MEDIUM	2021-08-30	功能优化	• 修复AOF（append-only file）持久化Rewrite时内存泄露的问题。 • 细分QPS（Queries Per Second）统计，当前支持统计读、写与其他，共计三类QPS。
1.7.5	MEDIUM	2021-08-16	功能优化	• 增强了槽（slot）的无感迁移可靠性，增强稳定性。
1.7.4	HIGH	2021-08-11	缺陷修复	• 修复集群实例开通直连的场景下，变配存在概率失败的问题。
1.7.2	MEDIUM	2021-07-27	功能优化	• 增强稳定性。
1.7.1	MEDIUM	2021-07-20	新特性	• 支持透明数据加密TDE（Transparent Data Encryption），可对RDB数据文件执行加密和解密。
			功能优化	• 优化了TairHash数据结构的内存占用。
1.6.15	MEDIUM	2021-07-12	功能优化	• 增强稳定性。
1.6.9	LOW	2021-06-22	新特性	• 支持延时统计直方图（Latency）。

小版本号	更新级别	发布日期	类型	说明
1.6.6	MEDIUM	2021-06-08	功能优化	• 增强稳定性。

1.6.3	LOW	2021-05-17	新特性	• 支持清空时按Key或Key pattern保留部分Key的能力，可用于：
				  ◦ 执行通过数据闪回按时间点恢复数据时，恢复部分Key或Key pattern。
				  ◦ 按照Key或Key pattern来删除或者保留keyspace的内容。

1.6.2	MEDIUM	2021-04-25	功能优化	• 优化在无感扩缩容场景下的性能和迁移速度。 • 支持通过公网获取虚拟IP（VIP）地址，为使用直连模式客户端提供更好的支持。 • 优化大Key格式。
1.6.1	MEDIUM	2021-04-08	新特性	• 默认开启大Key（big key）统计功能。 • TairString支持gt version。
			功能优化	• 提升迁移槽（slot）和无感扩容的能力。
1.5.1	HIGH	2021-03-15	缺陷修复	• 修复大Key（big key）实时统计在覆盖写入同名Key场景下的正确性问题。
1.5.0	MEDIUM	2021-02-22	新特性	• 支持大Key（big key）实时统计。
			功能优化	• 优化大内存场景下调用fork时，高可用系统HA（High Availability）的探活能力，避免可能出现的长时间停顿问题。
1.4.16	HIGH	2021-01-11	新特性	• 通过全球分布式缓存或DTS构建的多活链路在被释放后，可快速清理元数据，以便能够快速进行下次同步。更多信息，请参见全球分布式缓存和通过DTS实现Redis双向同步。
			缺陷修复	• 修复在使用TairHash场景下频繁执行**FLUSHALL**导致的内存泄露问题。
1.4.13	LOW	2020-11-27	新特性	• 当返回 `illegal address` 错误消息时，Redis会将当前客户端的IP地址包含在错误消息中。您可以根据提示，为Redis实例设置正确的IP白名单。 IP地址提示 
1.4.12	MEDIUM	2020-11-26	新特性	• TairString的语法中扩展FLAGS等标志，可兼容Memcache语义。
			功能优化	• 增强高可用系统HA（High Availability）的探活能力。 **说明** 建议升级到1.5.0小版本及以上，以获得针对此功能的最新优化。

小版本号	更新级别	发布日期	类型	说明
1.4.9	HIGH	2020-10-22	缺陷修复	• 修复TairString expire过期时间生成不正确binlog的问题，避免引起主备不一致。 • 修复TairHash在只读场景下仍在执行Active Expire，导致HA切换异常的问题。 • 修复强行停止实例后，重新启动时可能发生的崩溃问题。 • 修复通过数据闪回按时间点恢复数据模式下，加载RDB文件时会删除其中已过期Key的问题。
1.4.8	HIGH	2020-10-14	缺陷修复	• 修复部分模块加载时可能出现的内存泄露问题。
1.4.7	MEDIUM	2020-10-12	功能优化	• 将**CLUSTER NODES**命令的执行结果进行缓存以优化命令执行速度。
1.4.6	MEDIUM	2020-09-28	功能优化	• 优化部分模块在特殊场景下的处理能力。
1.4.1	MEDIUM	2020-09-08	新特性	• Proxy节点支持透传客户端IP地址至运行日志和审计日志，便于解读日志和定位具体的客户端。
			功能优化	• 优化数据采集能力，避免连接数过多且I/O线程繁忙场景下对数据分片的影响。
1.3.17	MEDIUM	2020-08-04	功能优化	• 优化使用DTS构建双向数据同步场景下的链路延，更多信息，请参见Redis企业版实例间的双向同步。
1.3.16	HIGH	2020-07-19	新特性	• 支持ECS安全组功能，通过为Redis实例绑定ECS所属安全组的方式实现快速授权（无需手动填写ECS的IP地址），可提升运维的便捷性。更多信息，请参见通过ECS安全组设置白名单。 • 更新TairString模块，支持更多兼容memcache语义的API（flags）。
			缺陷修复	• 修复在执行通过数据闪回按时间点恢复数据的场景下，**BGREWRITEAOF**被打断的问题。 • 订正审计日志中的latency标记位，避免其在主备审计日志中混淆。
1.3.9	MEDIUM	2020-06-19	功能优化	• 通过全球分布式缓存或DTS构建多向数据同步的场景下，执行清空数据操作时，支持自动清理保存的点位元数据信息，保障同步器（Replicator）的快速恢复。更多信息，请参见Redis全球多活简介和通过DTS实现Redis双向同步。
1.3.7	LOW	2020-05-19	新特性	• **INFO**命令返回值中，Replication部分支持展示role信息（例如 `role:master` ），可兼容Redisson客户端在部分场景下对该信息的调用。
1.3.6	MEDIUM	2020-05-19	新特性	• 定期将热Key（hot key）信息打印到日志中以便于查看。 • 性能指标的数据统计功能支持区分读、写及同步操作产生的QPS，统计更加精准。
			功能优化	• 优化通过数据闪回按时间点恢复数据的内核能力，简化数据恢复流程。 • **AUTH**、**ADMINAUTH**、**CONFIG**等命令不记录敏感信息，提升安全性。

小版本号	更新级别	发布日期	类型	说明
1.3.5	HIGH	2020-04-22	缺陷修复	• 修复多线程引擎在异步释放客户端连接时，可能产生的死锁问题。 • 修复引擎中文件描述符FD（File Descriptor）不能线性扩大的问题。
1.3.3	HIGH	2020-04-22	新特性	• TairBloom改用64位Hash算法。 • TairBloom新增对最终内存使用量的估算，用于精确内存统计。 • TairHash新增exhgetAll2接口，用于订正命令的响应格式。
			缺陷修复	• 修正未设置正确白名单时，Redis返回的错误提示，由 `(error) ERR invalid password` 修正为 `(error) ERR illegal address` 。 • 修复使用TairGIS操作多个POLYGON时可能出现的内存泄露问题。 • 修复TairDoc的默认路径问题。 • 修复Pub和Sub类命令在多线程引擎中，可能出现的竞争问题。

1.3.1	HIGH	2020-04-03	新特性	• 支持数据闪回功能，最长可恢复7天内任意时间点的Redis数据，避免误操作带来的数据损失，极大降低了运维复杂度，实时保护用户数据。更多信息，请参见通过数据闪回按时间点恢复数据。
				• TairGIS命令支持兼容Redis GEO相关命令。
				• TairBloom命令支持对BFRESERVE接口的容量安全校验。
				• TairHash命令支持更多新特性：
				  ◦ **EXHSET**、**EXHEXPIRE**、**EXHINCRBY**、**EXHINCRBYFLOAT**等命令增加了NOACTIVE选项，在某些场景下可降低内存开销。
				  ◦ **EXHINCRBY**增加了MAX和MIN选项，实现边界保证。
				  ◦ **EXHLEN** 增加了noexp选项，用于返回真实长度。
				• 支持Hash结构下的**HINCRBY**和**HINCRBYFLOAT**命令，可使用此命令事务性能力，对同一Key下的多个field做同增和同减操作。

			功能优化	• 优化数据结构模块的使用，更多信息，请参见数据结构模块集成。 • 大幅提升JedisCluster客户端在集群架构实例中，执行**MGET**和**MSET**的性能。
			缺陷修复	• 修复binlog占用空间溢出的问题。 • 修复热点Key在被执行逐出时可能出现的崩溃问题。 • 修复TairHash可能出现的double deallocation引发崩溃的问题。 • 修复关闭审计日志时因UAF（Use-After-Free）导致的崩溃问题。
1.0.10	LOW	2020-02-19	新特性	• 增加**BITFIELD_RO**命令，大幅优化其在读写分离场景下的性能。 **说明** 如果**BITFIELD**命令只有get选项，Proxy节点会将此命令转换为**BITFIELD_RO**并转发到后端多个数据分片上。

小版本号	更新级别	发布日期	类型	说明
1.0.9	HIGH	2020-02-19	缺陷修复	• 修复执行Lua脚本中的某些复杂命令时导致的复制进程崩溃的问题。
1.0.8	HIGH	2020-02-10	功能优化	• 优化流控的算法和性能。
			缺陷修复	• 修复由于客户端output buffer堆积触发服务端过载保护后，可能引起的服务崩溃问题。
1.0.5	HIGH	2020-02-01	新特性	• 支持全球多活功能，该功能是阿里云基于云数据库Redis自研的多活数据库系统，可轻松支持异地多个站点同时对外提供服务的业务场景，助力企业快速复制阿里巴巴异地多活架构。更多信息，参见Redis全球多活简介。 • 增加binlog功能及其协议，以支持全球多活等服务能力。
			缺陷修复	• 修复在使用直连模式时，INFO命令的返回信息可包含 `cluster_enabled` 信息，使某些SDK能够正确地自协商至集群模式。 • 修复管控客户端数量统计不准确的问题。 • 修复客户端释放时，可能出现的崩溃问题。 • 修复**pipeline**中包含复杂命令时，可能出现的崩溃问题。
0.2.9	HIGH	2020-01-06	功能优化	• 优化TairHash的内存使用量。
			缺陷修复	• 修复执行流控时可能出现的崩溃问题。
0.2.7	HIGH	2019-12-23	新特性	• 支持直连模式，客户端通过直连地址可绕过代理，与连接原生Redis集群的方法类似，可降低链路开销，进一步提升Redis服务的响应速度。更多信息，请参见开通直连访问。 • 支持热点Key的识别逻辑并优化内存占用，可精确地发现引擎的热点Key。 • 支持**EXCAS** 命令，可适用于乐观锁场景。使用方法，请参见EXCAS和降低乐观锁的性能消耗。
			缺陷修复	• 修复使用pipeline时可能出现的coredump问题。
0.2.3	LOW	2019-12-03	首次发布	Redis企业版性能增强型适合并发量大、读写热点多，对性能的要求极高的业务场景，相较于Redis社区版，优势如下： • 采用多线程模型，性能约为同规格社区版实例的3倍。 • 提供多种增强型数据结构模块（modules），包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom以及TairDoc，使业务无需再关心存储的结构和时效性，能够极大提升业务开发效率。

企业版（持久内存型）

小版本号	更新级别	发布日期	类型	说明
1.2.3.3	LOW	2022-05-12	新特性	• 新增支持TairCpc。

小版本号	更新级别	发布日期	类型	说明
1.2.3.2	HIGH	2022-04-24	缺陷修复	• 修复MAXMEMORY_VOLATILE_TTL模式下，淘汰Key时顺序异常的问题。 • 升级、优化集群架构增、删节点流程。 • 增强半同步的稳定性。
1.2.3.1	LOW	2022-03-31	功能优化	• 修复CAS命令持久化的问题。 • 修复集群版实例在慢日志和审计日志中不展示客户端IP地址的问题。 • 增强稳定性。
1.2.3	MEDIUM	2022-03-03	新特性	• 支持TairString。 • 支持Stream数据结构。 • Set结构支持intset结构，降低内存开销。 • 支持淘汰数据（Eviction policy）。 • 优化性能并增强稳定性。
			缺陷修复	• 修复实时Key分析功能遗漏统计Spop命令的问题。
1.2.2.4	LOW	2022-01-21	功能优化	• 针对大Value场景进行了存储空间优化，减少持久内存占用。
1.2.2.3	HIGH	2021-12-30	功能优化	• 优化写入数据时的长尾延迟（Tail Latency）问题。 • 增强稳定性。
1.2.2.2	HIGH	2021-12-14	缺陷修复	• 修复特殊场景下List、Hash等结构使用量统计错误的问题。
1.2.2.1	LOW	2021-10-21	功能优化	• 增强稳定性。
1.2.2	HIGH	2021-10-20	功能更新	• 优化集群版自动扩、缩容的速度控制。 **说明** 在扩、缩容的过程中会根据实例负载自适应地控制旧数据的清理速度，例如在低负载时会使用更多的CPU进行清理，加快清理速度。该功能会使得扩缩容过程中CPU使用率处于较高水位，但并不会影响正常访问。 • 优化大规模集群中CLUSTER NODES等命令的性能。
			缺陷修复	• 修复集群版扩、缩容时会生成异常慢日志（SLOWLOG）的问题。 • 增强稳定性。
1.2.0	LOW	2021-09-21	功能更新	• 针对List、Hash、Set与Zset结构进行了存储空间优化，减少内存和持久内存占用。
1.1.8	LOW	2021-08-17	功能更新	• 解决了部分场景下内存耗尽导致持久内存无法充分利用的问题。 • 增强稳定性。

小版本号	更新级别	发布日期	类型	说明
1.1.7	LOW	2021-08-02	新特性	• 支持半同步功能，默认未开启，如有需要可提交工单申请开启。 说明 半同步即表示客户端发起的更新在主节点执行完成后，会将日志同步传输到备节点，备节点确认接收后主节点才返回给客户端。当备实例不可用或者主备实例间出现网络异常时，半同步会退化为异步。 • INFO命令的返回信息中，redis_version返回值改为4.9.9，同时增加pena_version作为引擎版本（小版本）。
1.1.6.1	MEDIUM	2021-06-10	功能更新	• 增强稳定性。
1.1.6	MEDIUM	2021-05-08	新特性	• 支持无感扩缩容，可实现槽（slot）的无感迁移。
			功能优化	• Hash和Zset结构支持ziplist编码，降低内存开销。 • 增强高可用系统HA（High Availability）的探活能力。 • 优化大集群（即数据分片数量多）的迁移能力。 • 增强集群架构下的扩缩容的稳定性。
1.1.5	MEDIUM	2021-01-15	新特性	• 支持集群架构，可轻松突破Redis自身单线程瓶颈，满足大容量、高性能的业务需求。 • 使用实例的小版本作为INFO命令返回参数中redis_version的值。
			功能优化	• 增强服务稳定性。
1.1.4	MEDIUM	2020-10-28	新特性	• 普通账号执行INFO命令时，返回结果支持显示持久内存的使用信息。
			功能优化	• 优化EMBSTR编码方式占用的内存空间，降低内存开销。 • 增强服务稳定性。
1.1.3	LOW	2020-09-16	首次发布	Redis企业版（持久内存型）基于Intel 傲腾™数据中心级持久内存（AEP），为您提供大容量、兼容Redis的内存数据库产品，优势如下： • 相同容量下对比阿里云Redis社区版，价格降低30%左右。 • 解决大规格下执行AOF重写调用fork引起的延时抖动、服务数据加载慢等问题，无需在性能与持久化中取舍。 • 强大的命令级持久化保障，每个写操作持久化成功之后返回。 • 兼容原生Redis绝大部分的数据结构和接口，支持数据结构的持久化（除Streams以外）。 说明 更多关于命令限制的信息，请参见持久内存型命令限制。

企业版（容量存储型）

小版本号	更新级别	发布日期	类型	说明

小版本号	更新级别	发布日期	类型	说明
2.2.15	HIGH	2022-06-06	修复缺陷	• 增强稳定性。
2.1.13	LOW	2022-05-05	功能优化	• 优化主备同步的稳定性。
2.1.12	LOW	2022-04-26	功能优化	• 优化主备同步的稳定性。
2.1.10	LOW	2022-04-14	功能优化	• 优化主备同步效率，同步时吞吐更大，时延更低。 • 优化大Key集中过期时产生Compact任务的资源消耗。
2.1.7	LOW	2021-08-12	功能优化	• 优化主从切换（主动HA）速度，提升稳定性。
2.1.5	HIGH	2021-07-13	缺陷修复	• 修复Binlog日志文件占用空间过多的问题。
2.1.4	MEDIUM	2021-07-07	功能优化	• 优化TB级别的容量实例的存储参数以提升性能。 • 增强稳定性。
2.1.0	LOW	2021-05-26	新特性	• 支持事务（**MULTI**、**EXEC**命令）。 • 支持Lua脚本（**EVAL**、**EVALSHA**和**SCRIPT** 命令）。
2.0.13	LOW	2021-04-24	新特性	• 支持Keyspace Event通知。
1.2.17	HIGH	2021-02-04	缺陷修复	• 修复在极端情况下**FLUSHALL**可能造成的主备数据不一致的问题，增强主备数据一致性的保护能力。
1.2.16	HIGH	2021-01-29	缺陷修复	• 修复极端情况下主备连接断开后无法自动重连的问题，增强主备数据一致性的保护能力。
1.2.15	MEDIUM	2021-01-20	新特性	• 增加cmd_slowlog_count，支持统计slowlog历史累计值。
			功能优化	• 优化磁盘写满场景下的防护能力。
1.2.14	LOW	2020-12-31	新特性	• 增加主备数据校验。
1.2.13	HIGH	2020-12-31	缺陷修复	• 修复**RESTORE**命令在某些场景下的缺陷。
1.2.12	MEDIUM	2020-12-23	功能优化	• 优化**FLUSHALL**和**FLUSHDB**命令执行机制，限制其串行执行。
1.2.11	MEDIUM	2020-12-10	功能优化	• 优化内存管理。
1.2.10	LOW	2020-12-04	新特性	• 支持专有网络VPC免密功能，更多信息，请参见开启专有网络免密访问。 • 支持限制单次事务的大小（通过**max-write-batch-size**选项配置），避免因事务过大引起的内存溢出问题。
1.2.9	MEDIUM	2020-11-27	功能优化	• 为兼容Redis-shake工具执行数据迁移的场景，在**INFO**命令的返回信息中增加redis_version信息。
1.2.8	HIGH	2020-11-20	缺陷修复	• 修复连接数统计错误的问题。 • 修复**RESTORE**命令对Key统计错误的问题。

小版本号	更新级别	发布日期	类型	说明
1.2.7	HIGH	2020-10-28	缺陷修复	• 修复SET RANGE命令异常退出的问题。 • 修复Key为空字符串时导致的数据异常问题。
1.2.6	HIGH	2020-09-28	缺陷修复	• 修复连接管理日志增长过快的问题。
1.2.5	HIGH	2020-09-27	缺陷修复	• 修复运行日志增长过快的问题。
1.2.4	HIGH	2020-09-27	功能优化	• 增强主备同步的稳定性。 • 优化对复杂数据结构成员扫描的性能。
			缺陷修复	• 修复ZADD、ZINCR命令与原生命令不一致的问题（操作多个重复member时语义错误）。 • 修复MGET命令对空字符串返回 `nil` 的错误。
1.2.3	LOW	2020-09-27	首次发布	Redis企业版容量存储型基于TairDB KV存储引擎自主研发，采用ESSD云盘作为存储，可提供大容量、低成本、强持久化的数据库服务，优势如下： • 数据全部存储在云盘中，天然支持持久化，成本最低为Redis社区版的15%。 • 采用多IO多worker模式，提升单机吞吐能力，同时支持binlog格式的主备复制能力。 • 兼容大部分原生Redis命令。 • 解决了原生Redis固有的fork问题而预留部分内存的问题。

7.2.2. Redis社区版小版本发布日志

为提升用户体验，云数据库Redis会不定期地发布小版本，用于丰富云产品功能或修复已知缺陷。您可以参阅本文了解Redis社区版小版本的更新说明，选择在业务低峰期升级实例的小版本。

如何查询或升级实例的小版本

您可以通过控制台查看当前的小版本，具体操作及升级小版本的其注意事项，请参见升级小版本。

查看小版本

> 注意
> - 系统会自动检测实例的小版本，如果**小版本升级**按钮不存在或处于无法单击的状态，表示该实例已经是最新的小版本。
> - 由于各地域版本发布进度可能有所差异，小版本发布情况以当前控制台显示为准。

更新级别说明

- LOW：一般级别，包含日常新功能升级（例如新增某个功能）。
- MEDIUM：推荐级别， 包含功能模块优化类的升级（例如优化了某个功能）。除此以外，还包含了LOW级别所包含的更新内容。
- HIGH：重要级别，包含影响稳定性或安全性的重要升级（例如修复某个漏洞或缺陷）。除此以外，还包含LOW和MEDIUM级别

所包含的更新内容。

大版本为7.0版本

小版本号	更新级别	发布日期	类型	说明
7.0.0.4	LOW	2022-06-20	功能优化	• 更新至Redis开源社区7.0.2版本。
7.0.0.3	LOW	2022-05-27	首次发布	• 首次发布Redis社区版云盘版实例的7.0版本，更多信息请参见Redis 7.0 release notes。

大版本为6.0版本

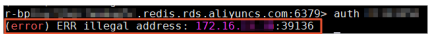
小版本号	更新级别	发布日期	类型	说明
0.1.19	LOW	2022-06-22	功能优化	• 优化延迟统计直方图，详情请参见延时洞察。
			安全加固	• 升级、优化集群架构增、删节点流程，详情请参见调整云盘实例的分片数量。
0.1.18	LOW	2022-05-17	功能优化	• 删除INFO命令返回的Errorstats - Selected字段。
0.1.17	LOW	2022-05-10	功能优化	• 集群架构支持MOVE命令。
0.1.16	MEDIUM	2022-04-25	安全加固	• 优化升级集群架构增、删节点流程，增强稳定性。
0.1.15	LOW	2022-03-24	新特性	• 支持延时统计直方图（Latency），详情请参见延时洞察。 • 支持INFO命令返回Errorstats（Redis 错误统计）信息。
0.1.14	LOW	2022-02-21	功能优化	• 支持READONLY、READWRITE命令，详情请参见Redis社区版命令支持。
0.1.13	LOW	2022-01-14	功能优化	• 增加DB元数据监控项：占用内存。 • 优化实时Top Key统计功能。
0.1.12	HIGH	2021-10-26	安全加固	• 修复集群变配过程中慢日志过多的问题，增强稳定性。
0.1.11	HIGH	2021-10-13	功能优化	• 增强了集群变配时槽（slot）无感迁移的自治能力。 • 优化CLUSTER NODES等命令的性能。 • 优化白名单功能。
0.1.10	HIGH	2021-09-06	功能优化	• 增强稳定性。
0.1.9	MEDIUM	2021-08-16	功能优化	• 增强了槽（slot）的无感迁移可靠性，增强稳定性。
0.1.8	MEDIUM	2021-08-06	功能优化	• 增强稳定性。
0.1.7	MEDIUM	2021-08-06	功能优化	• 增强稳定性。
			新特性	• 合并社区6.0.14版本的更新内容，更多信息，请参见Redis 6.0 release notes。

0.1.6 小版本号	MEDIUM 更新级别	2021-07-19 发布日期	类型	说明
			功能优化	• 优化Slot（槽）迁移数据完成后删除源端Slot数据的流程，增强数据可靠性。 • 简化Slot迁移数据的增量数据同步流程。
0.1.5	MEDIUM	2021-06-04	功能优化	• 增强稳定性。
0.1.4	MEDIUM	2021-05-27	功能优化	• 增强稳定性。
0.1.3	LOW	2021-05-18	新特性	• 支持大Key（big key）实时统计。
0.1.2	MEDIUM	2021-05-07	功能优化	• 增强稳定性。
0.1.1	MEDIUM	2020-11-28	新特性	• 合并社区6.0.9版本的更新内容，更多信息，请参见Redis 6.0 release notes。 • 支持槽（slot）的无感迁移能力。 • 支持通过公网获取虚拟IP（VIP）地址，为使用直连模式客户端提供更好的支持。
			功能优化	• 优化实例健康检查能力，提升在磁盘抖动场景下的主备切换速度。
0.0.5	HIGH	2020-08-21	缺陷修复	• 修复热点Key统计不准确的问题。
0.0.4	HIGH	2020-07-20	缺陷修复	• 修复部分参数的配置在重启后失效的问题。 • 修复慢日志对链路中备库标志错误的问题。
0.0.3	LOW	2020-06-11	新特性	• 合并社区6.0.5版本的更新内容，更多信息，请参见Redis 6.0 release notes。 • INFO命令返回值中，Replication部分支持展示role信息（例如 `role:master` ），可兼容Redisson客户端在部分场景下对该信息的调用。 • 支持对读写命令QPS的统计，更多信息，请参见查看监控数据。
0.0.2	LOW	2020-06-02	新特性	• 合并社区6.0.4版本的更新内容，更多信息，请参见Redis 6.0 release notes。
0.0.1	LOW	2020-05-06	首次发布	• 首次发布的小版本，基于社区6.0.1版本，更多信息，请参见Redis 6.0 release notes。

大版本为5.0版本

小版本号	更新级别	发布日期	类型	说明
5.1.9	LOW	2022-06-22	功能优化	• 优化延迟统计直方图，详情请参见延时洞察。
			安全加固	• 升级、优化集群架构增、删节点流程，详情请参见调整云盘实例的分片数量。
5.1.8	LOW	2022-05-17	功能优化	• 删除INFO命令返回的Errorstats - Selected字段。

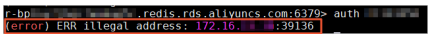
小版本号	更新级别	发布日期	类型	说明
5.1.7	LOW	2022-05-06	功能优化	• 集群架构支持MOVE命令。
5.1.6	MEDIUM	2022-04-25	功能优化	• 优化升级集群架构增、删节点流程，增强稳定性。
5.1.5	LOW	2022-04-13	功能优化	• 增强稳定性。
5.1.4	LOW	2022-03-24	新特性	• 支持延时统计直方图（Latency），详情请参见延时洞察。 • 支持INFO命令返回Errorstats（Redis 错误统计）信息。
5.1.3	LOW	2022-02-21	功能优化	• 支持READONLY、READWRITE命令，详情请参见Redis社区版命令支持。
5.1.1	LOW	2022-01-04	功能优化	• 增加DB元数据监控项：占用内存。
5.0.9	LOW	2021-12-22	功能优化	• 优化实时Top Key统计功能。
5.0.8	LOW	2021-11-15	功能优化	• 支持集群版无感扩、缩容。
0.5.17	HIGH	2022-06-07	安全加固	• 增强稳定性。
0.5.16	HIGH	2022-05-23	安全加固	• 提升集群架构实例开通直连访问后，变配的稳定性。
0.5.15	HIGH	2022-04-25	安全加固	• 修复集群架构实例开通直连访问后，变配过程中迁移大Key概率性失败的问题。
0.5.14	LOW	2022-01-04	功能优化	• 增加DB元数据监控项：占用内存。
0.5.12	LOW	2021-11-29	功能优化	• 修复异常情况下集群实例重启失败的问题。
0.5.11	LOW	2021-11-24	功能优化	• 修复实时Key分析功能遗漏统计Spop命令的问题。
0.5.10	HIGH	2021-10-26	安全加固	• 修复集群变配过程中慢日志过多的问题，增强稳定性。
0.5.9	HIGH	2021-10-15	安全加固	• 增强稳定性。
0.5.8	MEDIUM	2021-10-13	功能更新	• 增强了集群变配时槽（slot）无感迁移的自治能力。
0.5.7	LOW	2021-08-26	新特性	• 细分QPS（Queries Per Second）统计，当前支持统计读、写与其他，共计三类QPS。
0.5.6	HIGH	2021-08-16	缺陷修复	• 增强了槽（slot）的无感迁移可靠性，增强稳定性。
0.5.5	HIGH	2021-08-05	缺陷修复	• 修复集群实例开通直连的场景下，变配存在概率失败的问题。
0.5.4	MEDIUM	2021-07-27	功能更新	• 增强稳定性。
0.5.3	MEDIUM	2021-07-21	功能更新	• 优化数据迁移完成后删除源端数据的流程，增强数据可靠性。 • 简化数据迁移的增量数据同步流程。

小版本号	更新级别	发布日期	类型	说明
0.5.2	HIGH	2021-04-26	安全加固	• 主要解决社区Lua JIT的安全性漏洞。
			新特性	• 优化槽（slot）的迁移能力，云盘版实例可基于此实现无损扩缩容。 • 支持大Key（big key）实时统计。 • 支持通过公网获取虚拟IP（VIP）地址，为使用直连地址用户提供更好的支持。
0.5.0	MEDIUM	2021-03-25	新特性	• 支持槽（slot）的无感迁移能力。
			功能优化	• 增强在处理大量异步客户端请求场景下的稳定性。
0.4.0	MEDIUM	2021-03-09	新特性	• 支持大Key（big key）实时统计。 • 支持**CONFIG RESETSTAT**命令。 • 当返回 `illegal address` 错误消息时，Redis会将当前客户端的IP地址包含在错误消息中。您可以根据提示，为Redis实例设置正确的IP白名单。 IP地址提示 
			功能优化	• 优化实例健康检查能力，提升在磁盘抖动场景下的主备切换速度。
0.3.10	HIGH	2020-09-25	缺陷修复	• 修复**CLUSTER NODES**命令返回值与开源协议不一致的问题，即槽（slot）间以空格分隔，避免客户端解析错误。
0.3.9	LOW	2020-07-20	新特性	• 支持ECS安全组功能，通过为Redis实例绑定ECS所属安全组的方式实现快速授权（无需手动填写ECS的IP地址），可提升运维的便捷性。更多信息，请参见通过ECS安全组设置白名单。
0.3.8	HIGH	2020-07-14	功能优化	• 开放**CLIENT UNBLOCK**子命令。
			缺陷修复	• 修复在迁移slot（槽）时解析expire（过期时间）不正确的问题。 • 订正审计日志中的latency标记位，避免其在主备审计日志中混淆。
0.3.7	HIGH	2020-06-17	缺陷修复	• 修复直连模式下返回IP地址不可达的问题。
0.3.6	LOW	2020-06-09	新特性	• **INFO**命令返回值中，Replication部分支持展示role信息（例如 `role:master` ），可兼容Redisson客户端在部分场景下对该信息的调用。
0.3.5	LOW	2020-06-05	新特性	• 支持对读写命令QPS的统计，更多信息，请参见查看监控数据。
0.3.4	HIGH	2020-04-08	缺陷修复	• 修复热点Key在被执行逐出时可能出现的崩溃问题。 • 修复关闭审计日志时因UAF（Use-After-Free）导致的崩溃问题。

小版本号	更新级别	发布日期	类型	说明
0.3.1	HIGH	2020-02-20	新特性	• 支持审计日志功能，为您提供日志的查询、在线分析、导出等功能，更多信息，请参见开通新版审计日志。 • 支持直连模式，客户端通过直连地址可绕过代理，连接方式类似连接原生Redis集群，可降低链路开销，进一步提升Redis服务的响应速度。更多信息，请参见开通直连访问。 • 支持在专有网络VPC免密访问的场景下，申请公网连接地址。 • 增加**INFO**命令中关于 `oom_err_count` 的统计场景：即maxmemory数据量超过设定值时会被计入。
			缺陷修复	• 修复当**RPOPLPUSH**命令的source和destination相同时，在执行过程中因触发了过期机制导致的崩溃问题。 • 修复专有网络VPC免密模式下权限验证错误的问题。
0.2.0	LOW	2020-01-17	新特性	• 支持实时热点Key统计，帮助您快速发现实例中的热点Key，更多信息，请参见实时Top Key统计。
0.1.2	LOW	2019-11-26	新特性	• 支持在读写分离实例的只读节点中执行只读的lua脚本。
0.1.2之前	不涉及	不涉及	不涉及	• 5.0版本下的早期小版本，建议升级至最新的小版本。

大版本为4.0版本

小版本号	更新级别	发布日期	类型	说明
1.9.12	HIGH	2022-06-07	安全加固	• 增强稳定性。
1.9.10	HIGH	2022-05-23	安全加固	• 提升集群架构实例开通直连访问后，变配的稳定性。
1.9.9	HIGH	2022-04-25	安全加固	• 增强稳定性。
1.9.8	HIGH	2022-04-25	安全加固	• 修复集群架构实例开通直连访问后，变配过程中迁移大Key概率性失败的问题。
1.9.6	HIGH	2021-10-15	安全加固	• 增强稳定性。
1.9.5	LOW	2021-09-13	新特性	• 细分QPS（Queries Per Second）统计，当前支持统计读、写与其他，共计三类QPS。
1.9.4	HIGH	2021-08-05	缺陷修复	• 修复集群实例开通直连的场景下，变配存在概率失败的问题。
1.9.3	MEDIUM	2021-07-20	功能更新	• 增强稳定性。
1.9.2	HIGH	2021-04-19	安全加固	• 主要解决社区Lua JIT的安全性漏洞。
			新特性	• 支持通过公网获取虚拟IP（VIP）地址，为使用直连地址用户提供更好的支持。

小版本号	更新级别	发布日期	类型	说明
1.9.1	MEDIUM	2021-03-08	功能优化	• 优化实例健康检查能力，提升在磁盘抖动场景下的主备切换速度。 • 优化大内存实例通过fork执行**BGSAVE**和**REWRITE**的能力，避免可能出现的长时间停顿问题。
1.9.0	LOW	2021-02-22	新特性	• 当返回 `illegal address` 错误消息时，Redis会将当前客户端的IP地址包含在错误消息中。您可以根据提示，为Redis实例设置正确的IP白名单。 IP地址提示 
1.8.8	HIGH	2020-09-25	缺陷修复	• 修复**CLUSTER NODES**命令返回值与开源协议不一致的问题，即槽（slot）间以空格分隔，避免客户端解析错误。
1.8.7	LOW	2020-07-20	新特性	• 支持ECS安全组功能，通过为Redis实例绑定ECS所属安全组的方式实现快速授权（无需手动填写ECS的IP地址），可提升运维的便捷性。更多信息，请参见通过ECS安全组设置白名单。
1.8.6	HIGH	2020-07-14	缺陷修复	• 订正审计日志中的latency标记位，避免其在主备审计日志中混淆。
1.8.5	LOW	2020-06-09	新特性	• **INFO**命令返回值中，Replication部分支持展示role信息（例如 `role:master` ），可兼容Redisson客户端在部分场景下对该信息的调用。
1.8.4	LOW	2020-06-05	新特性	• 支持对读写命令QPS的统计，更多信息，请参见查看监控数据。
1.8.3	HIGH	2020-04-08	缺陷修复	• 修复热点Key在被执行逐出时可能出现的崩溃问题。 • 修复关闭审计日志时因UAF（Use-After-Free）导致的崩溃问题。
1.8.1	LOW	2020-02-20	新特性	• 支持在已开启专有网络VPC免密访问的场景下，申请公网连接地址的操作。
1.8.0	HIGH	2020-01-16	新特性	• 支持实时热点Key统计，帮助您快速发现实例中的热点Key，更多信息，请参见实时Top Key统计。
			缺陷修复	• 修复在直连访问场景下，**INFO**命令的返回信息可包含cluster_enabled信息，使某些SDK能够正确地自协商至集群模式。
1.7.1	MEDIUM	2019-11-20	新特性	• 支持在读写分离实例的只读节点中执行只读的lua脚本。 • 支持直连模式，客户端通过直连地址可绕过代理，像连接原生Redis集群一样连接阿里云Redis集群，可降低链路开销，进一步提升Redis服务的响应速度。更多信息，请参见开通直连访问。 • **INFO**命令的返回信息中，Memory部分中增加对lua脚本内存量的统计。
			功能优化	• 优化审计日志的对内存使用。
1.5.8	HIGH	2019-09-23	缺陷修复	• 修复旧版全球多活链路中，双向同步时**SET EX**的原子性被破坏的问题。

小版本号	更新级别	发布日期	类型	说明
1.5.6	HIGH	2019-08-28	新特性	• 审计日志支持记录latency事件。
			缺陷修复	• 修复客户端发起的KEYS、FLUSHALL、FLUSHDB等命令引发的慢请求可能引起主备切换的问题。
1.5.4	LOW	2019-07-08	新特性	• 支持审计日志功能，为您提供日志的查询、在线分析、导出等功能，更多信息，请参见开通新版审计日志。 • 支持对整个事件循环的latency记录，帮助您了解引擎的状态。
1.5.2	HIGH	2019-07-04	缺陷修复	• 修复当RPOPLPUSH命令的source和destination相同时，在执行过程中因触发了过期机制导致的崩溃问题。
1.4.0	HIGH	2019-05-15	缺陷修复	• 修复实例重启后，进入加载RDB或AOF的状态会触发主备切换的问题。
1.4.0之前	不涉及	不涉及	不涉及	• 4.0版本下的早期小版本，建议升级至最新的小版本。

7.2.3. Proxy小版本发布日志

为提升用户体验，云数据库Redis会不定期地发布Proxy（代理）节点的小版本，用于丰富云产品功能或修复已知缺陷。您可以参阅本文了解Proxy小版本的更新说明，选择在业务低峰期升级小版本。

如何查询或升级Proxy的小版本

您可以通过控制台查看当前的小版本，具体操作及升级小版本的其注意事项，请参见升级小版本。

查看Proxy小版本

> **说明**
> - 系统会自动检测实例的小版本，如果**集群代理升级**按钮不存在或处于无法单击的状态，表示小版本已经是最新。
> - 由于各地域版本发布进度可能有所差异，小版本发布情况以当前控制台显示为准。

Proxy介绍

在云数据库Redis的集群架构和读写分离架构中，代理服务器（Proxy）承担着路由转发、负载均衡与故障转移等职责。通过了解Proxy的路由转发规则和特定命令的处理方式，有助于您设计更高效的业务系统。更多信息，请参见Redis Proxy特性说明。

更新级别说明

- LOW：一般级别，包含日常新功能升级（例如新增某个功能）。
- MEDIUM：推荐级别， 包含功能模块优化类的升级（例如优化了某个功能）。除此以外，还包含了LOW级别所包含的更新内容。
- HIGH：重要级别，包含影响稳定性或安全性的重要升级（例如修复某个漏洞或缺陷）。除此以外，还包含LOW和MEDIUM级别所包含的更新内容。

6.8.x

小版本号	更新级别	发布日期	类型	说明
6.8.2	MEDIUM	2022-06-14	功能优化	• 增强稳定性，修复一些Crash问题。
6.8.1	LOW	2022-04-19	新特性	• 支持部分TairSearch。 • 支持TairRoaringV2.2新增的命令。
6.8.0	MEDIUM	2022-04-01	新特性	• 支持部分TairZset。 • 支持部分TairRoaring。 • SSL证书禁用RC4加密算法。
			缺陷修复	• 修复开启ptod_enabled参数后，可能导致SDIFFSTORE、SINTERSTORE、SUNIONSTORE、ZINTERSTORE、ZUNIONSTORE命令异常的问题。 • 修复SMOVE命令可能出现CROSSSLOT的错误。

6.7.x

小版本号	更新级别	发布日期	类型	说明
6.7.9	MEDIUM	2022-03-05	缺陷修复	• 修复DBSIZE、KEYS命令在部分Redis节点异常时，返回的Response中结尾的 `\n` 被截断的问题。
6.7.8	MEDIUM	2022-03-03	缺陷修复	• 禁用SCRIPT DEBUG命令。 • 修复ZINTERSTORE、ZUNIONSTORE生成数据的score精度只有6位小数的问题。 • 修复Redis 5.0部分架构下SDIFF、SDIFFSTORE命令返回结果错误的问题。
6.7.7	LOW	2022-01-30	功能优化	• 增强稳定性。
6.7.6	LOW	2022-01-20	功能优化	• 增强稳定性。
6.7.5	MEDIUM	2022-01-10	功能优化	• 优化 `RANDOMKEY` 命令随机获取不同的Redis节点，避免多次 `RANDOMKEY` 命令落在同一个Redis节点。
			缺陷修复	• 修复info Commandstats对性能增强版实例聚合结果错误的问题。
6.7.4	MEDIUM	2021-12-20	功能优化	• 增强稳定性。
6.7.3	MEDIUM	2021-12-15	缺陷修复	• 修复SSL连接时，首次请求存在概率不响应的问题。
6.7.2	LOW	2021-11-30	功能优化	• 增强稳定性。
6.7.1	MEDIUM	2021-11-23	功能优化	• 增强稳定性。

6.6.x

小版本号	更新级别	发布日期	类型	说明
6.6.14	MEDIUM	2021-11-01	功能优化	• 修复ECS架构下（split_multi_key_cmd_as_slot开启），ZINTERSTORE、ZUNIONSTORE存在概率不返回的问题。

小版本号	更新级别	发布日期	类型	说明
6.6.13	MEDIUM	2021-10-22	功能优化	• 修复开启Proxy Query Cache后，热升级存在概率失败的问题。
6.6.12	MEDIUM	2021-10-12	功能优化	• 增强稳定性。
6.6.11	MEDIUM	2021-10-11	功能优化	• 增强稳定性。
6.6.10	MEDIUM	2021-09-27	缺陷修复	• 修复Memcache实例在只读或只写请求下返回消息错误的问题。
6.6.9	MEDIUM	2021-09-06	缺陷修复	• 修复CVE-2021-3711漏洞与CVE-2021-3712漏洞。
6.6.8	MEDIUM	2021-08-30	功能优化	• 增强稳定性。
6.6.7	MEDIUM	2021-08-27	功能优化	• 修复开启Statistics功能后内存泄露的问题。
6.6.6	LOW	2021-08-13	功能优化	• 增强稳定性。
6.6.5	LOW	2021-08-03	新特性	• 支持Memcache Gateway模式，即可实现Memcache协议的支持和转发。
6.6.4	HIGH	2021-07-08	新特性	• CLIENT LIST、CLIENT KILL命令支持展示和操作进程维度的连接。
			缺陷修复	• 修复TairZset命令不支持大写的问题，更多信息，请参见TairZset。
6.6.3	MEDIUM	2021-06-18	功能优化	• 优化多可用区容灾场景下的内部管控。
6.6.2	LOW	2021-06-08	新特性	• 增加对部分内部命令的支持。
6.6.1	LOW	2021-05-26	新特性	• 新增TairZset数据结构，实现任意维度的double类型的分值排序，提升数据处理效率，且客户端适配简易，无需任何编解码封装。更多信息，请参见TairZset。
6.6.0	LOW	2021-04-28	新特性	• 新增代理查询缓存功能（Proxy Query Cache），启用后代理节点会缓存热点Key对应的请求和查询结果，当在有效时间内收到同样的请求时直接返回结果至客户端，无需和后端的数据分片交互，可更好地改善对热点Key的发起大量读请求导致的访问倾斜。更多信息，请参见通过Proxy Query Cache优化热点Key问题。

6.5.x

小版本号	更新级别	发布日期	类型	说明
6.5.9	HIGH	2021-04-21	缺陷修复	• 修复特殊场景下，多Key命令死循环的问题。
6.5.8	HIGH	2021-04-16	缺陷修复	• 本版本为特殊版本，即在6.5.5小版本基础上，修复在选择多个DB的场景下，请求乱序的问题。
6.5.7	HIGH	2021-04-16	缺陷修复	• 修复在选择多个DB的场景下，请求乱序的问题。

小版本号	更新级别	发布日期	类型	说明
6.5.6	MEDIUM	2021-04-09	新特性	• SCAN命令支持的最大数据分片数由256提升为1024。 • 当订阅的Channel（频道）所在的Slot（槽）发生迁移后，Proxy会断开订阅的连接让客户端重连以保障数据一致性。
			功能优化	优化Proxy命令处理机制： • 处理MOVED命令时，将请求重新发给MOVED的地址。 • 发送不带Key的命令时，屏蔽Slot为空的数据分片。
6.5.5	HIGH	2021-03-05	缺陷修复	• 修复在主备切换或变更配置而触发DHT信息更新时，可能导致的内存泄露问题。
6.5.4	HIGH	2021-02-07	缺陷修复	• 修复客户端接收返回信息过慢可能出现的内存泄露问题。
6.5.3	HIGH	2021-01-21	新特性	• 支持在Lua脚本中的KEYS下标中使用变量。
			缺陷修复	• 修复集群架构下，数据分片超过32个时使用MULTI或BLOCK类命令引发的内存泄露问题。
6.5.2	HIGH	2021-01-19	缺陷修复	• 修复alb enat模式下通过Socket获取虚拟IP地址（VIP）地址失败的问题。
6.5.1	LOW	2021-01-14	新特性	• 慢日志在记录多Key命令相关日志时，支持记录最后返回Response的数据分片的IP地址。
6.5.0	HIGH	2020-12-24	缺陷修复	• 修复执行GIS.GET ALL命令可能导致的崩溃问题。关于该命令的详细介绍，请参见TairGIS。

6.4.x

小版本号	更新级别	发布日期	类型	说明
6.4.10	MEDIUM	2020-12-01	功能优化	• 优化密码错误场景下的提示信息，易于理解。
6.4.9	HIGH	2020-11-06	缺陷修复	• 修复多线程模式下开启SSL加密功能导致的崩溃问题。 • 修复执行UNSUBSCRIBE时，Channel（频道）中包含0时导致的Response协议错误的问题。
6.4.8	HIGH	2020-10-21	功能优化	• 运行日志对大包、ASK回复包和MOVED包的二进制请求进行编码后记录，避免日志乱码问题。
			缺陷修复	• 修复max_session_processing（单个连接允许堆积的最大请求数）的配置不能被动态设置的问题。更多参数的介绍，请参见参数支持。
6.4.7	MEDIUM	2020-10-09	功能优化	• 优化Proxy节点的内部监控。
6.4.6	HIGH	2020-09-30	缺陷修复	• 修复因节点角色未初始化，导致的标准或集群架构的实例执行SLOWLOG命令可能超时的问题。 • 修复了特定规格的Memcache实例通过数据管理DMS连接失败的问题。 • 修复订阅 `__keyspace@0__` 时，未指定Key导致的崩溃问题。

小版本号	更新级别	发布日期	类型	说明
6.4.5	LOW	2020-09-27	新特性	• 增加对部分内部命令的支持。
6.4.3	HIGH	2020-09-25	功能优化	• 针对Jedis客户端中pipeline的特殊实现进行了适配，优化连接限制的释放计算，Jedis连接示例，请参见Jedis客户端。
			缺陷修复	• 修复**BZPOPMIN**和**XREAD**命令错误记录了慢日志的问题，更多信息，请参见查询慢日志。
6.4.2	HIGH	2020-09-09	缺陷修复	• 修复空闲连接默认1分钟后被断开的问题。
6.4.1	MEDIUM	2020-08-25	新特性	• 新增Timeout配置，空闲的客户端连接会被自动断开。 • 支持统计只读节点上的慢日志信息，即**SLOWLOG**命令会发送至所有Master节点和只读节点。更多信息，请参见查询慢日志。
			功能优化	• 优化了PubSub和Monitor连接的内存使用，避免因内存碎片引起的内存快速上涨。 • 提升了Proxy节点处理新连接的能力。
6.4.0	HIGH	2020-08-18	缺陷修复	• 修复ConfigServer在完成配置前调用stat导致的崩溃问题。

6.3.x

小版本号	更新级别	发布日期	类型	说明
6.3.9	MEDIUM	2020-08-14	新特性	• 慢日志支持记录真实的客户端IP地址，帮助您更好地定位慢日志，更多信息，请参见查询慢日志。
			功能优化	• 提升了Proxy节点的短连接处理能力。
6.3.8	HIGH	2020-07-24	缺陷修复	• 修复Vector Clear不释放内存导致的内存上涨的问题。 • 添加cpc命令支持
6.3.7	HIGH	2020-07-13	缺陷修复	• 修复开启SSL加密功能后，建立连接时可能出现的崩溃问题。
6.3.5	HIGH	2020-07-10	新特性	• 为审计日志中的二进制数据执行编码，提升日志易读性。 • 增加**no_loose_statistics-ip-enable**、**no_loose_statistics-keys**、no_loose_statistics-cmds参数，可实现对IP、Key和命令维度的统计，更多详细介绍请参见参数支持。
			缺陷修复	• 修复连接被释放后，执行 `CheckExceedLimitAndClose` 可能导致的崩溃问题。 • 修复SSL加密功能开启失败的问题。
6.3.4	HIGH	2020-05-21	缺陷修复	• 修复 `\r\n` 等空包可能导致后续请求不返回的问题。

8.基本概念

本文为您介绍云数据库Redis版的基本概念。

术语	说明
Redis	云数据库Redis版是一款依据BSD开源协议发行的高性能Key-Value存储系统（Cache and Store）。
实例ID	每个实例对应一个用户空间，实例是使用云数据库Redis版的基本单位。云数据库Redis版对单个实例根据不同的容量规格有不同的连接数、带宽、CPU处理能力等限制。用户可在控制台中看到自己购买的实例ID列表。
标准版双副本实例	指一主节点一副本架构的云数据库Redis版实例。标准版双副本实例能扩展的容量和性能有限，但可以变配到其它架构，例如集群版或读写分离版。
集群实例	指具有集群扩展性的云数据库Redis版实例。集群实例有更好的扩展性和性能，但是在功能上也有一定的限制。
连接地址	用于连接云数据库Redis版的Host地址，以域名方式展示，可在**实例信息 > 连接信息**中查询到。
连接密码	用于连接云数据库Redis版的密码。密码拼接方法为 `实例ID:自定义密码` 。例如，在购买时设置的密码为 `1234` ，分配的实例ID为 `xxxx` ，那么密码即为 `xxxx:1234` 。
逐出策略	与Redis的逐出策略保持一致，详情请参见：Using Redis as an LRU cache。
DB	即Redis中的Database。云数据库Redis版支持256个DB：DB 0至DB 255，默认写入到第0个DB中，无法修改总DB数。